Elektrisk Mätteknik, LTH

Datorbaserade mätsystem

3. Bussystem

Lund 06-09-06

INNEHÅLL

W. Cornish, IEEE-488
General Purpose Instrumentation
Bus Manual,

Chapter 1

The General Purpose Instrumentation Bus

INTRODUCTION

This book provides a complete description of the fundamentals of the IEEE-488 interface system, which is commonly referred to as the General Purpose Instrumentation Bus (GPIB). It will provide full technical details, written in easy-to-understand language, to the first-time system user or technician, as well as be a valuable reference to the professional GPIB system designer or programmer.

The IEEE-488 interface system has been carefully designed to provide an integration of one or more instruments to a computer or controller, which allows two-way communication, to simplify and automate the testing of any electric, electronic, or electromechanical device in production, now or in the future. Since it is a design philosophy as well as an interface system, it greatly simplifies the role of the test engineer in the design and implementation of automatic test equipment (ATE). It has been designed with the necessary flexibility to accommodate a growing number of all types of electronic instruments which are being manufactured today and tomorrow.

The IEEE-488 interface is a digital system in which up to 15 instruments or devices may communicate with each other, under control of a master unit, when connected together in parallel using specially designed cables and connectors referred to as the "bus." Supervision of the system is provided by a master unit, called the controller, which is usually either a common personal computer or a dedicated bus controller. The software that is required by the system can be easily written by anyone who is familiar with the BASIC computer language, or is obtainable from many sources which provide a multitude of universal programming packages.

Since modern-day IEEE-488 bus controllers provide the necessary protocol, an IEEE-488 system can be set up and run using a few user-friendly software commands. This allows a first-time user to be able to design a simple system in little more time than it would take with a manual test setup. These basic commands, as well as the more complex, are explained in detail in the chapters to follow.

HISTORY

In September 1965 the Hewlett-Packard Company, Palo Alto, California, began to investigate the possibility of interfacing any and all of its future instruments with each other. It was obvious that the level of sophistication of electronics technology was advancing at such a rate that more complex and superior test instrumentation would soon be commonplace. As new instruments were developed, it became clear that in many test situations it would be almost impossible to train sufficient numbers of production personnel to properly operate these instruments to their fullest capability. As each new type of test instrument or device was added to the arsenal available to the test engineer, the number of tests which could be performed, and their complexity, increased at an exponential rate.

In order to partially eliminate the human element in the ever more complex test setups, some form of communication was needed so that one instrument could "talk" to another and vice versa. That was the beginning of what was originally called the Hewlett-Packard interface bus, more commonly referred to as HP-IB.

A newly formed group called the International Electrotechnical Commission (IEC) took the initial efforts of Hewlett-Packard Company as a starting proposal for an interface system. In September 1974 this proposal was approved for balloting by the IEC. In April 1975 the Institute of Electrical and Electronic Engineers (IEEE) published a document known as IEEE-488/1975, entitled "Digital Interface for Programmable Instrumentation." This contained the electrical, mechanical, and functional specifications of an American standard interface system. In January 1976 the American National Standards Institute (ANSI) published an identical standard called MC1.1.

In November 1978 the IEEE-488 document was revised, primarily for editorial classification and addendum, and the new document was identified as IEEE 488-1978. This document has been the standard for the general-purpose instrumentation bus (GPIB) which has been adopted by hundreds of manufacturers all over the world. More than 40,000 copies of the IEEE-488 document have been distributed to more than 250 manufacturers in 14 or more countries. It is estimated that there are more than 4000 products which use the GPIB byte serial, bit parallel interface system for automatic or semiautomatic testing.

IEEE-488.2

When the IEEE-488 standard was adopted in 1975, designers and users were able to use the system, but not without many problems which they were forced to solve. Because the IEEE-488 document had purposely left some problems unsolved, it was up to the users to determine which instruments would function with the controller, and each other, as required. It soon became apparent that each manufacturer handled message protocol and data handling differently.

A first attempt to standardize the data formats resulted in the creation of a document called "IEEE 728, Recommended Practice for Code and Format Conventions for use With IEEE 488-1978." The formats which were recommended had evolved over time and had worked well with the interface system. This helped provide the information which would later be included in the next major revision of the IEEE-488 document.

In June 1987 the IEEE approved a new standard for programmable instruments and devices. This was called IEEE Standard 488.2-1987 Codes, Formats, Protocols, and Common Commands. The original document, IEEE 488-1978, was retitled IEEE-488.1 The new standard works with, and enhances, the original one. Some of the issues which IEEE-488.2 addresses are:

1. A required minimum set of IEEE-488.1 capabilities
2. Reliable transfer of messages between a talker and listener
3. Precise syntax in those messages
4. A set of commands which would be useful in all instruments
5. Common serial poll status reporting
6. Synchronizing programming with instrument functions
7. Automatic address assignments

The IEEE-488.2 standard was designed to make the interface system easier to use by requiring that all devices provide certain capabilities such as talk and listen, respond to device clear commands, and be capable of service requests. Although other functions such as parallel poll and device trigger are left optional with the instrument manufacturer, IEEE-488.2 requires that when these functions are implemented, they provide a minimum capability level.

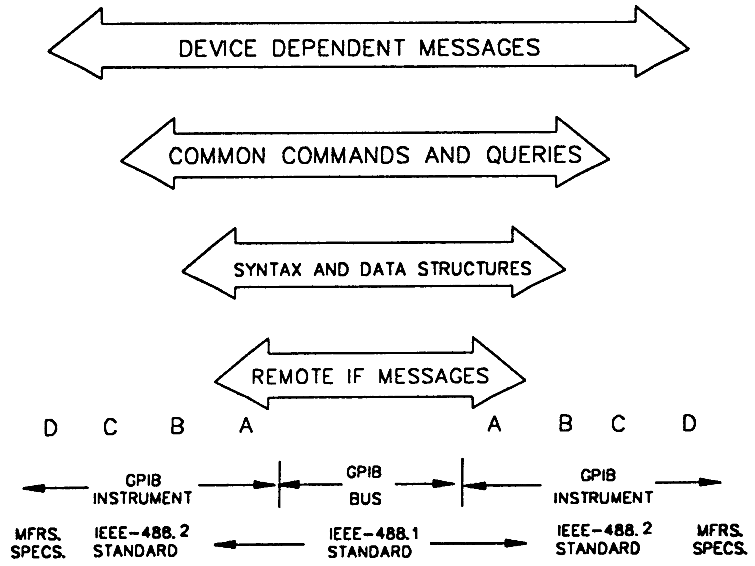


Figure 1-1. GPIB structure illustrating the IEEE-488.1 and 2 standards.

Figure 1-1 illustrates how the IEEE-488.2 standard works with the existing standard IEEE-488.1. The interface system can be divided into several functional layers, with the lowest layer being the Remote Interface Messages layer or the IEEE-488.1 bus. The function of this section includes the mechanical aspects such as the cable and connectors, the electrical specifications, and the handshake function.

The next two layers are defined by the new IEEE-488.2 standard. These consist of the Syntax and Data Structures layer and the Common Commands and Queries layer. The Syntax and Data Structures layer defines how data is transmitted between devices by specifying the usage of the ASCII (American Standard Code for Information Exchange) character set for data representation. Included in this specification is the data format for binary numbers.

The uppermost layer is the Device Dependent Messages layer which is defined by the manufacturer of the instrument or device. The messages which are defined here are also called device commands; they control the performance and functions of the GPIB device in accordance with the requirements of the device as specified by the manufacturer.

MAJOR INTERFACE SYSTEMS

There are now four major standards of interface systems. They are:

1. IEEE 488-1978 (now called IEEE 488.1)
2. ANSI MC1.1 (identical to IEEE 488.1)
3. IEC 625.1 (identical except for the connector)
4. B.S. 6146 (British standard, identical to IEC 625.1)

Today the most widely used interface system is the IEEE-488, and it is implemented in several brand versions: HP-IB, GPIB, IEEE Bus, ASCII Bus, and Plus Bus. For all practical purposes IEEE-488, HP-IB, and GPIB are used synonymously. This book will use these terms interchangeably, and they can be considered to be identical.

Today, all manufacturers of test equipment must provide GPIB capability to remain competitive. Any instrument which does not contain this capability is limited in its application and may be noncompetitive in the vast world of electronic test instrumentation. This must be considered by test engineers and those who are involved with production testing of electronic or electromechanical instruments and devices. The use of an interface system must be considered at the onset of a testing program to increase efficiency, reduce costs, and help eliminate errors on the production line.

DESCRIPTION OF THE IEEE-488 (GPIB) INTERFACE SYSTEM

In the GPIB interface system three categories of instruments or devices are used. These are talkers, listeners, and controllers. A talker is a unit which is able to transmit on the bus pertinent measurement data or information concerning its status, either asynchronously or in response to a command from the controller. There can be only one active talker on the bus at any given time. Some examples of talkers are voltmeters, frequency counters, and tape readers. Talkers are generally listeners as well. A listener is a device which can receive commands and data when addressed and may or may not be capable of the talk function. There can be up to 14 active listeners simultaneously on the bus. Examples of listener devices which usually have no talk capability are printers, display devices, and programmable power supplies.

The controller is the brains of the system; it provides the commands (through its programming and software) that cause each and every instrument and device on the bus to perform its task. The controller is usually both a talker and listener. The ubiquitous personal computer can perform very satisfactorily as an IEEE-488 controller if it is so equipped. There are several dedicated IEEE-488 controllers, one of which is the Fluke model 1722A Instrument Controller.

Any instrument or device on the bus can be both a talker and listener (but not simultaneously). In any given interface system there can be only one active controller, but it is possible for a very complex interface system to have several controllers. In such cases one of these is specified as the master controller and it will determine which unit will be in control of the bus at any given time.

It is not necessary for an interface system to have a controller. A minimum GPIB system consists of a single talker and single listener, with no controller. An example of such a system would be

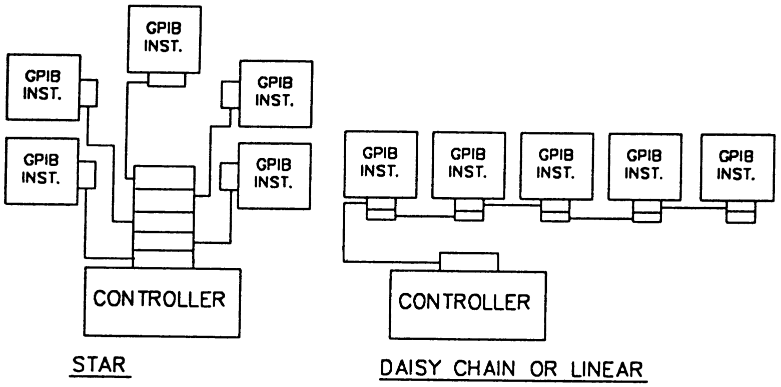


Figure 1-2. Two methods of interconnecting GPIB instruments into a system.

a voltmeter (talker) and a printer (listener). This system would periodically assimilate voltage readings and provide hard copy of them.

All instruments on the interface bus are connected in parallel to each other by means of an IEEE-488 cable containing 16 active wires that are terminated at each end in a specially designed hermaphrodite connector. This allows a group of instruments to be connected together in either a "star" or "daisy chain" configuration as illustrated in Figure 1-2.

Part of the IEEE-488 document includes the specifications for the connectors which must be used to interconnect all devices and instruments. Figure 1-3 illustrates the pin-out diagram of this ribbon-type connector, which contains 24 pins and constructed so that it contains both a male and female connector, similar to the arrangement which may be found on Christmas tree light sets. The purpose of this arrangement is to allow connectors to be stacked on top of each other so that in a given test setup all devices can be physically located in close proximity to each other. The male/female design of the connectors also permits interconnection of all units in the daisy chain or linear configuration. In any test setup, there will always be at least one connector available which can be used to add an additional instrument to the setup, if necessary.

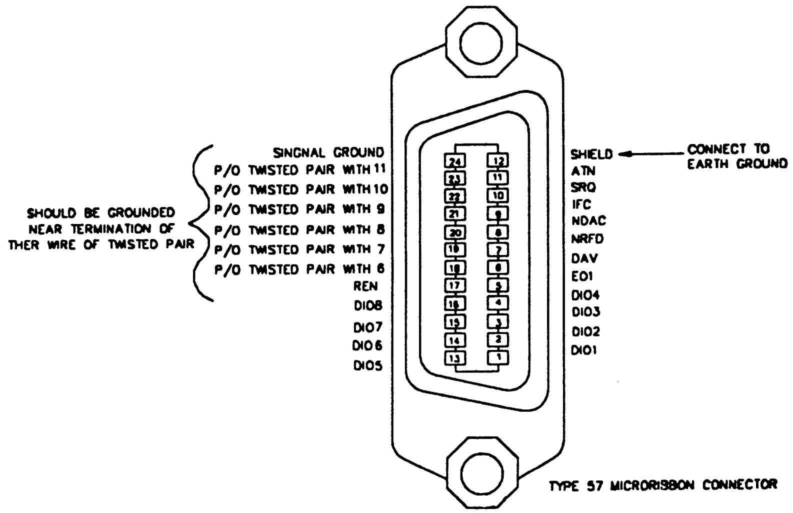


Figure 1-3. IEEE 488 connector showing the identity of the pin connections.

The IEEE-488 specification permits up to 15 devices to be connected together in any given setup, including the controller if it is part of the system. The maximum length of the bus network is limited to 20 meters total transmission path length. It is recommended that the bus be loaded with at least one instrument or device every 2 meter lengths of cable. If, under certain conditions, it is necessary to exceed the maximum permitted length of 20 meters, this limit may be increased by the use of IEEE-488 extenders. These devices contain active circuitry which can handle the added capacitance and inductance of long IEEE-488 cable lengths.

The cable, or "bus," which connects all instruments of the interface system in parallel with each other contains 16 active wires. Of these, eight are used for data transmission in a bit parallel, byte serial format. The remaining eight wires provide interface and

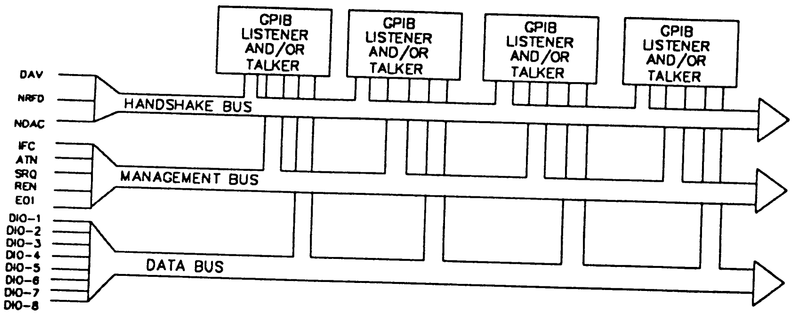

GPIB LISTENER AND/OR TALKER
GPIB LISTENER AND/OR TALKER
GPIB LISTENER AND/OR TALKER
GPIB LISTENER AND/OR TALKER
DAV
NRFD
NDAC
HANDSHAKE BUS
IFC
ATN
SRQ
REN
EOI
MANAGEMENT BUS
DIO-1
DIO-2
DIO-3
DIO-4
DIO-5
DIO-6
DIO-7
DIO-8
DATA BUS

Figure 1-4. General purpose instrumentation bus structure.

communication management. The bus is a two way communications channel, and data flows in both directions. Figure 1-4 illustrates the structure of the IEEE-488 bus and identifies the 16 connections of the interconnecting cable.

GPIB ADDRESSING

Since the bus is a "party line" type of communications channel, each instrument must be assigned a unique address so that any message or data transmitted by any device on the bus is accepted by only its intended recipient. A total of 31 addresses, called primary addresses, are available and these are usually selected for each instrument or device by means of a set of switches or jumpers located at the rear or inside the device. Figure 1-5 illustrates a typical set of address switches which usually may be found at the rear of an instrument which has IEEE-488 capability.

The address, 0 through 30, to which an instrument is set is determined by the decimal equivalent of the 5 binary bits represented by the switch or jumper positions. The switches or jumpers have a weight of 1, 2, 4, 8, and 16; therefore, the sum of the weights

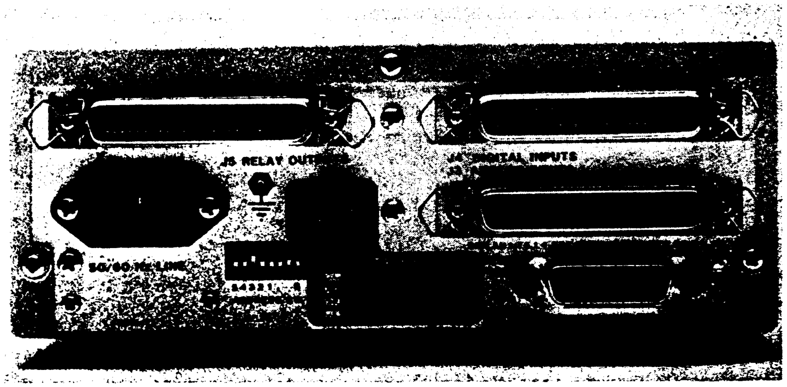


Figure 1-5. Typical rear panel of GPIB instrument. Courtesy of ICS Electronics Corp.

of all selected switches or jumpers becomes the GPIB address of the unit. Any address number from 0 to 30 is valid; address number 31 is reserved for control purposes and must not be used. Address 21 is usually reserved for the controller talk/listen address and is not recommended to be used for an instrument address. Every instrument or device which is part of a given interface system must be assigned its own unique address number and this cannot be shared by any other instrument on the bus.

Some instruments or devices may also require a secondary address which provides access further into the unit itself. Secondary addresses are usually preset at the factory, but may be changed in the field by rewiring a set of jumpers inside the device. It is permissible to duplicate secondary addresses on two or more instruments which are part of the same interface system. GPIB devices which are capable of accepting a secondary address command are called extended listeners and/or talkers.

The primary address which is selected by the user will actually specify two corresponding address codes on the data lines. These are called the talk address and listen address, and the sixth and seventh bits of the data byte are used to distinguish between the two. Figure 1-6 is an IEEE-488 code chart which illustrates the talk and listen ASCII address characters for all valid address codes.

It is usually not necessary to specify the individual talk or listen characters when programming commands, since most controllers in use today will automatically configure the sixth and seventh bits. A simple command such as OUTPUT 703 in HP BASIC, for example, will instruct the device with primary address 03 to listen. The command ENTER 703 will instruct that same device to talk. In this example, 7 as part of the address code is required by a typical Hewlett-Packard controller in accordance with a select code which can be physically set on an interface card located within the computer or plug-in interface. Select codes can come into play when there is more than one controller in the interface system.

Some GPIB devices (such as plotter/printers) may have more than one talk or listen address (multiple addresses), and these devices typically use fewer than the usual five bits to select the address. For example, if the bit 1 (binary weight) switch was omitted on a device, a single setting of the remaining four switches would select two addresses which are consecutive, such as 8 and 9 or 14 and 15.

Do not confuse multiple addresses with a secondary address. Each multiple address in a device is a primary address and must be treated as such in the software commands. To access a secondary address in a device the controller must first transmit the primary address, then the secondary, usually separated by a colon or comma.

GPIB COMMUNICATIONS

The GPIB is a two-way communications channel which carries data and bus management information on a total of 16 wires, and it is organized using 8 binary bits grouped into bytes which are called words. Data and address information is carried on the data input/output lines which are identified as DIO-1 through DIO-8. Of the remaining eight wires, three are used for the handshake function and the last five provide system control and management.

The GPIB uses a negative logic system which specifies a zero logic level on any line which is "true." Conversely, any line which

GPIB ADDRESS	SWITCH NO.					ADDR. CHAR.	
	5	4	3	2	1	TALK	LISTEN
00	0	0	0	0	0	@	SP
01	0	0	0	0	1	A	!
02	0	0	0	1	0	B	"
03	0	0	0	1	1	C	#
04	0	0	1	0	0	D	$
05	0	0	1	0	1	E	%
06	0	0	1	1	0	F	&
07	0	0	1	1	1	G	'
08	0	1	0	0	0	H	(
09	0	1	0	0	1	I	)
10	0	1	0	1	0	J	*
11	0	1	0	1	1	K	+
12	0	1	1	0	0	L	,
13	0	1	1	0	1	M	–
14	0	1	1	1	0	N	.
15	0	1	1	1	1	O	/
16	1	0	0	0	0	P	0
17	1	0	0	0	1	Q	1
18	1	0	0	1	0	R	2
19	1	0	0	1	1	S	3
20	1	0	1	0	0	T	4
21	1	0	1	0	1	U	5
22	1	0	1	1	0	V	6
23	1	0	1	1	1	W	7
24	1	1	0	0	0	X	8
25	1	1	0	0	1	Y	9
26	1	1	0	1	0	Z	:
27	1	1	0	1	1	[	;
28	1	1	1	0	0	\	<
29	1	1	1	0	1	]	=
30	1	1	1	1	0	^	>
31	1	1	1	1	1	–	?

Figure 1-6. GPIB address chart showing the address switch positions required for each address, and the ASCII characters for the TALK and LISTEN addresses.

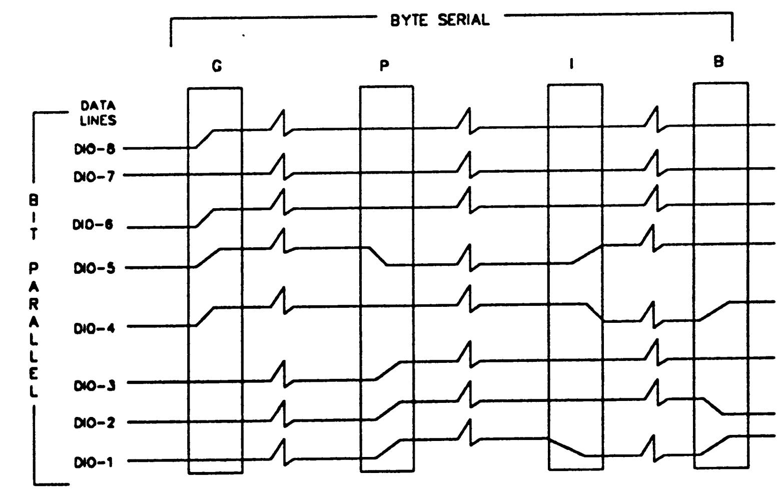
is open has a logic level of 1 and is defined as "not true," or "false." An important reason for this negative logic convention is to allow GPIB devices to be designed with transistor open collector output circuits which pull the lines to zero voltage level to indicate a true condition. Using this system, it is possible to connect all devices on the bus in parallel, and any one is thus capable of creating a true condition by its open collector transistor driver. Additionally, the negative logic convention reduces noise susceptibility in the true state and provides a not true, or logic 1, state on any line which is not in use or is disconnected.

Messages and data are asynchronously transferred over the bus in a byte serial, bit parallel format using an interlocking three-wire handshake technique. This ensures data integrity in a multiple listener system where the data acceptance of each listener can take place at widely different rates. A two-wire handshake system could allow multiple acceptance of the same ASCII character in the fast listener as the slower listeners were still assimilating data.

The maximum data transfer rate is 1 megabyte per second over limited distances. Using the full transmission path limits the rate to about 250 or 500 kilobytes per second. However, since the system is always limited by the acceptance rate of the slowest addressed listener, the actual data transmission rate in any given system may be much less.

Data is transferred from device to device over the bus using the eight bidirectional data lines. Normally a 7-bit ASCII code is used. The international equivalent to this is the 7-bit International Standards Organization (ISO) code. Figure 1-7 illustrates the byte serial, bit parallel sequence when transmitting the message "GPIB."

With various software techniques, other methods to compress information on the data lines may be employed, which will greatly enhance the speed with which data may be transferred. Some types of data, such as transfer of oscilloscope waveform information, require enormous amounts of information. More efficient methods of transmitting this data will improve the operating characteristics of the bus.



ASCII/ISO	G		P		I		B
DECIMAL	71		80		73		66

Figure 1-7. "GPIB" sequence as transmitted in bit parallel, byte serial format.

HANDSHAKE

The purpose of the handshake function in the GPIB interface system is to ensure that all messages transmitted on the bus are properly received by the addressed listeners. Many of the command messages which are used are intended to be received by more than one listener, and before any new messages can be initiated, each listener must acknowledge that it has properly received the message addressed to it. The GPIB system uses a three-line handshake, which was selected over previous systems that used just two lines.

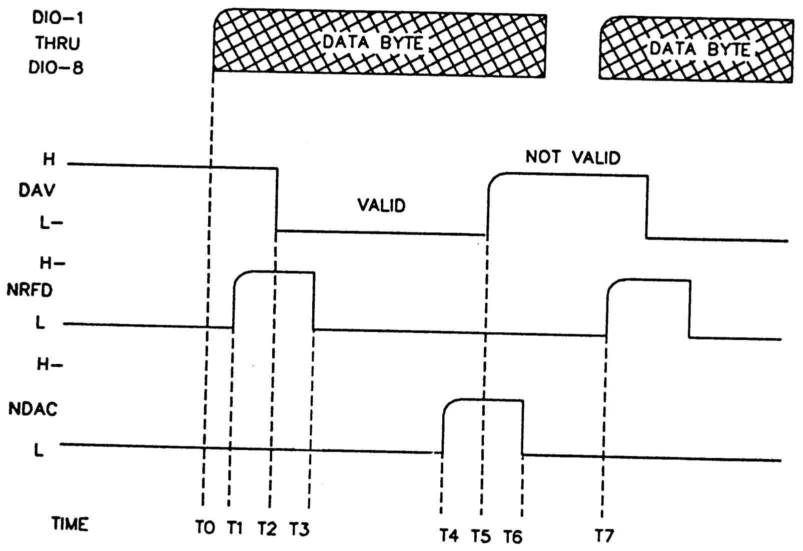


Figure 1-8. Timing diagram of the handshake sequence.

A three-wire system prevents multiple acceptance of data by a fast listener while a slow one is busy accepting the message.

Figure 1-8 illustrates the functions of the handshake lines as a data byte is transmitted and accepted by all addressed listeners on the bus. The three handshake lines which control the data transfer are:

1. NRFD (not ready for data). This line is used by all devices on the bus to indicate their conditions of readiness to accept a data transmission. If any instrument is not ready to receive data for any reason, it will pull the NRFD line low, indicating a true condition and inhibiting the controller (or any other talker on the bus) from transmitting. Only when all devices on the bus release this line so that its logic level is 1 will a data transmission be permitted.

2. DAV (data valid). This line is used by the source of data (talker) to indicate that the eight data lines are settled and valid. The DAV line does not become true (logic level zero) until the NRFD line has been released by all devices and allowed to assume a logic 1 level.
3. NDAC (no data accepted). The addressed listener(s) on the bus will set the NDAC line low (true) to indicate that it has not yet accepted the data. When the data is accepted, the NDAC line is released. If more than one listener is required to accept data, each will hold the line low until it does. When the slowest listener on the bus has finally accepted the data, the NDAC line is released and goes to the logic 1, or false, condition.

The following is a summary of the handshake sequence as illustrated in Figure 1-8, shown from time zero (T0) to T7:

- T0. The source (talker) checks for listeners and places a data byte on the DIO 1 through DIO 8 lines.
- T1. As all listeners become ready for the data, each releases the NRFD line so that it goes high with the slowest listener.
- T2. The source validates the data by pulling the DAV line low.
- T3. The first acceptor sets the NRFD line low, indicating that it is not ready for the next byte to follow.
- T4. When the slowest acceptor receives and accepts the data, it releases the NDAC line to indicate that all listeners on the bus have accepted the data.
- T5. The DAV line goes high to indicate that the data is no longer valid. The data may then change to form the next byte.
- T6. The first acceptor sets the NDAC line low in preparation to receive the next valid byte.
- T7. NRFD goes high with the slowest acceptor and the cycle is repeated.

When sending addressed commands, the interface message consists of the primary address of the intended acceptor, followed by the interface function. During this sequence the attention line is true so that the listeners accept the information on the bus as either an address or command.

The interface message can also contain a third element, called a device dependent command, which programs the receiving instrument to assume a certain condition or perform a certain function. For example, a voltmeter could be programmed to read dc volts or a frequency counter could be told to take a frequency reading.

Many typical commands to instruments and devices contain a group of characters and digits called a string. The acceptor receives the command, character by character, using the handshake function to acknowledge that it has properly received what was sent.

Chapter 2

IEEE-488 Protocol

In the IEEE-488 interface system there is a bidirectional flow of commands and data between the controller and interconnected devices. Communications between these units is achieved by sending and receiving a series of messages via the 16 lines of the bus. There are two basic categories of messages: interface and device dependent.

Interface messages are used to manage the bus and are called commands. They instruct the listeners and talkers in the system to assume a desired mode. These commands are used to initialize the bus, for setting devices to remote or local operation, to instruct devices to listen, unlisten, talk, untalk, and for other functions which may be required.

DEVICE-DEPENDENT MESSAGE UNITS

Device-dependent message units (sometimes referred to as data) contain the information that a GPIB device is to transmit on the bus and are not commands. For example, a voltmeter may have stored a reading in its buffer, and a command from the controller would instruct the meter to transmit it so that it can be stored in the controller's memory for further processing. Data assimilated by the controller could be sent to a printer which would then produce a hard copy. A digitizing oscilloscope may have stored the shape of a complex waveform to be later reproduced on the controller CRT. In short, device dependent message units are the data which is transmitted on the eight data I/O lines of the interface bus when the ATN line is false and the active talker is sourcing data to all active listeners.

Normally a 7-bit ASCII code (Figure 2-1) is used, but the manufacturer of a GPIB device is free to use any other encoding technique to compress information on the eight lines. Even when such standard formats as pure binary or BCD are used, the sequence between the least and most significant bits could be different for two manufacturers.

An amendment to the IEEE-488 standard, called IEEE-P981, may provide some standardization among the various manufacturers of GPIB instruments. This amendment establishes a common message structure and defines control protocol procedure.

INTERFACE FUNCTIONS

The IEEE-488 document specifies a total of 11 interface functions which can be implemented in any GPIB device. It is not necessary for all to be designed into an instrument; the manufacturer of that device is free to use as many or few as needed for the device to perform its intended function. Each interface function is identified by a mnemonic, which is a one- to three-letter word used to describe a particular capability. A brief description of each function is described in Table 2-1, and is covered in detail in Chapter 5.

In addition to the 11 basic interface capabilities illustrated in Table 2-1, the IEEE-488 document also describes in detail subsets of all functions. Each subset is identified by assigning a number (from zero up) after the mnemonic. For example, there are nine subsets of Talker (T) which are identified as T0 through T8. Subsets of interface functions are discussed in Chapter 5.

ASCII/ISO & IEEE CODE CHART

B7 B6 B5	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
BITS B4 B3 B2 B1	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
0 0 0 0	NUL	DLE	SP	0	@	P	`	p
0 0 0 1	SOH	DC1	!	1	A	Q	a	q
0 0 1 0	STX	DC2	"	2	B	R	b	r
0 0 1 1	ETX	DC3	#	3	C	S	c	s
0 1 0 0	EOT	DC4	$	4	D	T	d	t
0 1 0 1	ENQ	NAK	%	5	E	U	e	u
0 1 1 0	ACK	SYN	&	6	F	V	f	v
0 1 1 1	BEL	ETB	'	7	G	W	g	w
1 0 0 0	BS	CAN	(	8	H	X	h	x
1 0 0 1	HT	EM	)	9	I	Y	i	y
1 0 1 0	LF	SUB	*	:	J	Z	j	z
1 0 1 1	VT	ESC	+	:	K	[	k	{
1 1 0 0	FF	FS	,	<	L	\	l	:
1 1 0 1	CR	GS	−	=	M	]	m	}
1 1 1 0	SO	RS	.	>	N	^	n	~
1 1 1 1	SI	US	/	?	O	–	o	RUBOUT (DEL)
	ADDRESSED COMMANDS	UNIVER. COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

Figure 2-1. ASCII code chart illustrating values for bits 1 through 7 of a data byte.

INTERFACE MANAGEMENT LINES

In addition to the handshake lines there is a total of five general interface management lines which are used to provide an orderly flow of commands and data through the interface. These lines are:

Table 2-1. Interface Functions.

Interface Function	Mnemonic	Description
Talker (Extended Talker)	T (TE)	Device must be able to transmit
Listener (Extended Listener)	L (LE)	Device must receive commands and data
Source Handshake	SH	Device must properly transfer a multiline message
Acceptor Handshake	AH	Device must properly receive remote multiline messages
Remote/Local	RL	Device must be able to operate from front panel and remote information from bus
Service Request	SR	Device can asynchronously request service from the controller
Parallel Poll	PP	Upon controller request device must uniquely identify itself if it requires service
Device Clear	DC	Device can be initialized to a pre-determined state
Device Trigger	DT	A device function can be initiated by the talker on the bus
Controller	C	Device can send addresses, universal commands, address commands, and conduct polls
Drivers	E	This code describes the type of electrical drivers in a device

Attention, Interface Clear, Remote Enable, Service Request, and End Or Identify. The mnemonic identification for these lines and a brief description of each is as follows:

1. ATN. This line must be monitored by all devices on the bus and respond to it within 200 nanoseconds. Its purpose is to place the interface in the command mode when true, and

data mode when not true, or false. When the bus is in the command mode, all devices with listening capability must receive the next transmission and accept information on the data lines as either commands or addresses. When data is to be sent to addressed listeners, the ATN line is set high (not true).
2. IFC. This line is used only by the system controller to initialize the interface to a standby or idle state in which there is no activity on the bus. All devices must monitor this line at all times and respond to it within 100 microseconds. When IFC is set true, all devices addressed to either talk or listen are set to untalk or unlisten, and the serial poll function (if in use) is disabled.
3. REN. This line is used only by the system controller to place all listeners in the remote programming mode when they are addressed to listen. All devices capable of both local and remote operation must monitor the REN line and respond to it within 100 microseconds. When the system controller sets this line high (not true), all devices return to local operation.
4. SRQ. The Service Request line is used by one or more devices on the bus to indicate a need for attention, as might be caused by a syntax error, overload, etc. Such a request by any device can interrupt the current sequence of events. The SRQ line can be cleared by a serial poll only, and the controller must perform a poll of all devices on the bus to determine which one requires service.
5. EOI. When the ATN line is not true, the End or Identify line is used by an active talker to indicate the last byte of a data message. The IEEE-488 standard also permits the end of a transmission to be indicated by the generation of a line feed character (ASCII 10). When ATN is true, the controller uses this line to execute a parallel poll in which up to eight devices on the bus place a status bit on the eight data I/O lines.

UNIVERSAL COMMANDS

The five multiline universal commands are Device Clear, Local Lockout, Serial Poll Enable, Serial Poll Disable, and Parallel Poll Unconfigure. Unlike the four uniline commands already described (IFC, REN, ATN, IDY), these universal commands are transmitted on the data lines. Devices on the bus interpret such data as commands, since the ATN line is true. The universal commands are:

1. Device Clear (DCL). This command causes each recognizing device on the bus to return to a predefined state. The devices respond only if they are addressed or in remote control. The state at which each device is reset is defined by the manufacturer of the equipment.
2. Local Lockout (LLO). This command is used to disable the front panel controls or return to local pushbutton on those devices which recognize the command. This is useful during an automatic test sequence when it is mandatory that test personnel be prevented from altering a predetermined operating mode of the instrument or device.
3. Serial Poll Enable (SPE). This command causes all talkers on the bus to assume a serial poll mode. When each is addressed to talk the device will place on the bus data lines a single 8-bit word which contains information on the status of the device, as specified by the manufacturer of the equipment.
4. Serial Poll Disable (SPD). This command will follow the serial poll enable command so that each talker on the bus is returned to its normal state of outputting data when addressed to talk.
5. Parallel Poll Unconfigure (PPU). This command will reset all devices which recognize a parallel poll command so that they are in an idle state and unable to respond to it.

In addition to the five universal multiline commands described above, there are two additional universal commands which are technically classified as addresses. These are Untalk (UNT) and Unlisten (UNL).

The Untalk command unaddresses the current talker on the bus. It is not necessary to use this command to unaddress the current talker since addressing the next talker automatically unaddresses all others, but this command has been provided for convenience.

The Unlisten command unaddresses all listeners on the bus. It is not possible to unaddress one listener; all will be unaddressed. This command will precede addressing desired listeners when it is necessary that only such listeners receive the next data to be sent on the bus.

ADDRESSED COMMANDS

The following is a description of a group of commands which are called addressed commands. It is not necessary for devices on the bus to respond to any or all such commands, since the manufacturer of a GPIB instrument or device determines which, if any, are necessary for the proper operation of the unit when under remote control.

1. Group Execute Trigger (GET). This command causes all currently addressed devices which have GET capability to receive and respond to the command by initiating a preprogrammed action. For example. a voltmeter can take a reading, or a generator can produce a burst of oscillation. Some devices on the bus may also require an additional command to produce the desired function. The purpose of the GET command is to provide a trigger command which

can produce simultaneous triggering in all addressed devices.

2. Selected Device Clear (SDC). This command causes a currently addressed listener to reset to a predetermined state as specified by the manufacturer of the device.
3. Go to Local (GTL). This command causes the currently addressed listener to leave its remote state and return to manual front panel control. Should the device be readdressed with a subsequent command, it will return to the remote mode.
4. Parallel Poll Configure (PPC). This command is used in conjunction with a secondary command, Parallel Poll Enable Command. It causes the addressed listener to be configured in accordance with the PPE command which follows it. When the device receives the PPC command from the controller, it responds on a particular data I/O line to indicate its status. Another secondary command, Parallel Poll Disable (PPD), prevents any response from the the addressed devices that have received the PPC command.

POLLING

The interface system provides two methods of interrogating listeners by the controller. These are referred to as parallel poll and serial poll. It provides the GPIB programmer with two methods of determining the status of the devices on the bus.

Serial polling is performed in the form of a sequence in which each device on the bus is individually addressed and directed to return a status byte to indicate its condition. The controller should be directed to poll every device on the bus to be sure that each SRQ requestor is found. When the serial poll is completed, the controller then must transmit the Serial Poll Disable and Untalk commands so that each device on the bus is returned to the normal remote state.

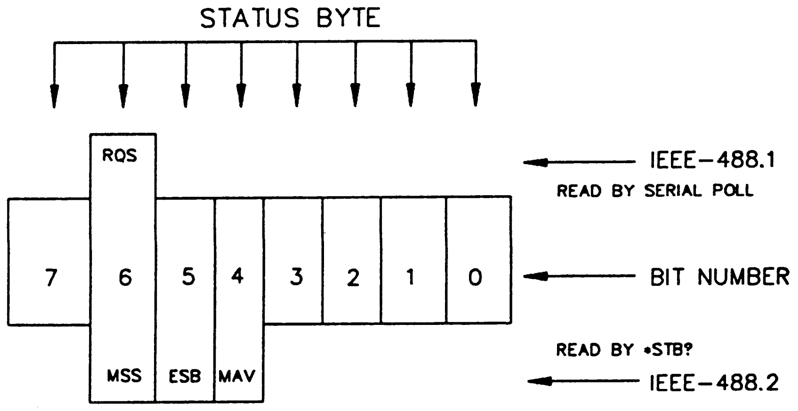


Figure 2-2. Serial poll status byte as defined by IEEE-488.1 and IEEE-488.2.

The advantage of the serial pole sequence is that it will return to the controller the identity of the requestor at the same time as its status byte is received. However, when there are many devices on the bus (all devices should be polled), this can be a very time consuming procedure.

IEEE-488.1 originally provided the serial poll to allow controllers to read the status of the instruments on the bus, but other than bit 6, RSQ, it left the definition of the remaining bits to the manufacturer of the device. IEEE-488.2 further defines two more bits of the status byte (Figure 2-2).

Bit 4 is defined as the message available bit (MAV). This indicates whether or not the output queue of the instrument is empty. When data is available, bit 4 will be true.

Bit 5 is the event status bit (ESB), which is used to indicate if an enabled standard event has occurred. Such events include (but are not limited to) power on, user request, command errors, and execution errors.

A faster method of polling is the parallel poll, but only eight devices can be polled at a time and just 1 bit of information from each can be transmitted to the controller. If there are more than eight devices on the bus which must be polled, this may be done in two or more steps. When the parallel poll sequence is initiated by the controller, up to eight designated devices on the bus will return its status bit on one of the data I/O lines. Each device can be directed to respond on a particular data line through the direction of the secondary command, PPE, which follows the Parallel Poll Configure command. It is also possible for some devices to be hard wired so that they will always respond on a designated data line.

It is possible to have more than one device respond on a given data line since the open collector output circuits of the devices allow a parallel connection with a resulting AND or OR of the status bits.

MESSAGE TERMINATION

In the IEEE-488 interface system there is no stipulation as to how long a message may be, so there must be some method by which all devices on the bus can determine that a message is indeed terminated. There are three methods which may be employed to accomplish this task.

The device which is talking on the bus may add the printer formatting line feed character (ASCII 10) to the end of each message. Another method is through the use of the dedicated End or Identify line (EOI) contained in the interface bus itself. This line has two purposes, End or Identify, and the ATN line is used to distinguish between each. When ATN is false, the EOI line can be used by the active talker to indicate to the listener(s) that the last byte of a data message has occurred. The third method allows the EOI line to be asserted concurrently with the line feed character.

Since there is no guarantee that any particular listener on the bus will respond to all types of message terminations, some manufacturers of GPIB devices have designed into their units the capability of choosing the type of message termination that best services the system. Table 2-2 is a representative illustration of the types of terminator commands which can be programmed into a Fluke digital meter. To implement any of these terminators, the meter would be instructed to do so when it is addressed by the controller.

Table 2-2. Typical Programmable Terminator Commands in a GPIB Device.

Command Syntax	Terminator
W0 (default)	Enable CR, LF, EOI
W1	Enable CR & LF only
W2	Enable CR & EOI only
W3	Enable CR only
W4	Enable LF & EOI only
W5	Enable LF only
W6	Enable EOI only
W7	Disable all output terminators

It is possible to create certain errors in a GPIB program if care is not taken to ascertain that the message terminator selected by the software engineer is compatible with all units on the bus. In the case in which data is transferred by binary block communications, it is possible that a data byte could be configured as the ASCII line feed character 00001010. Should this occur in a system in which a listener is programmed to respond to LF, it will automatically terminate the message and dump any following data bytes.

Another problem may occur when a carriage return precedes the line feed character. A controller may end the transmission when it receives CR, leaving the unsent LF character in the talker. Then, when that device is instructed to talk again, it will send the leftover LF character as the first data byte, creating an error in the controller.

This possible source of confusion over the message terminators has been resolved with IEEE-P981 amendment, which does not permit GPIB instruments to generate carriage returns for any reason.

Chapter 3

The IEEE-488.2 Standard

OVERVIEW

The IEEE-488.2 revision was adopted 10 years after the GPIB had been in use and was designed to help eliminate many of the user problems which plagued the GPIB. Some of the problems encountered were:

1. Device interface capabilities which would vary from one manufacturer's unit to another, even though the type of instrument might be identical. For example, one digital voltmeter might be a listen-only device which could not report back to the controller its readings. Another might not be fully programmable — for example, it might require a manual setup of its function or range.
2. Data formats might be totally different from one unit to another. One might communicate in ASCII while another might require binary or BCD coding. It would be difficult to substitute such units for each other in a given test setup

without performing some time-consuming rewriting of software.

3. Message protocol was not standardized. The order in which a unit receives or transmits commands and data was strictly up to the designer of the instrument.
4. There was a wide variation in status reporting between units from each manufacturer. Since the function of most of the 8 bits of the status byte was left up to the discretion of the manufacturer, it would not be possible to replace one unit with another without first modifying the test programming.
5. Device-dependent commands were different even though two identical units performed the same function. The software engineer could not assume that any command, no matter how common or frequent it might be used, would be suitable for all similar type instruments.

These problems, and others, were solved for the most part by the adoption of the IEEE Standard 488.2 Codes, Formats, Protocols, and Common Commands for Use with ANSI/IEEE Standard 488.1 — 1987. Included in the new standard is a set of codes, data formats, message protocols, and common commands which would be used with, and in addition to, the original standard (now identified as IEEE-488.1). Figure 3-1 illlustrates the structure of the new standard as it enhances the original one.

INTERFACE CAPABILITIES

IEEE-488.2 defines a minimum set of capabilities which each instrument or device must implement. Table 3-1 is a tabulation of these required capabilities. A listing of code definitions of the interface capabilities tabulated are in Appendix D.

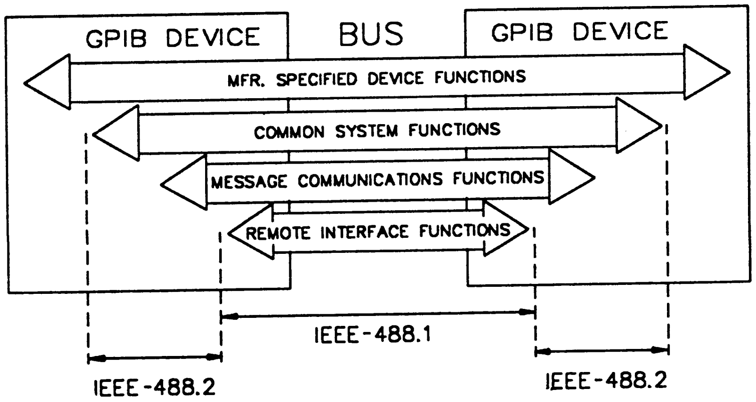

GPIB DEVICE
BUS
GPIB DEVICE
MFR. SPECIFIED DEVICE FUNCTIONS
COMMON SYSTEM FUNCTIONS
MESSAGE COMMUNICATIONS FUNCTIONS
REMOTE INTERFACE FUNCTIONS
IEEE-488.1
IEEE-488.2
IEEE-488.2

Figure 3-1 IEEE-488.1 and IEEE-488.2 functional structure.

SYNTAX AND DATA FORMATS

The IEEE-488.2 document specifies the required data format for any type of message that may be sent, including numbers and strings of characters. Included in the types of numbers which are specified are binary, octal, and hexadecimal. A listing of these

Table 3-1 Minimum IEEE-488.2 capabilities.

Capability	Code	Comment
Source Handshake	SH1	Full Capability
Acceptor Handshake	AH1	Full Capability
Talker	T(TE)5, or T(TE)6	Basic Talker, Serial Poll, untalk on MLA
Listener	L(LE)3, or L(LE)4	Basic Listener, Unlisten on MTA
Service Request	SR1	Full Capability
Device Clear	DC1	Full Capability
Remote Local	RLO or RL1	None or Full Capability
Parallel Poll	PPO or RL1	None or Full Capability
Device Trigger	DTO or DT1	None or Full Capability
Controller	CO or C4 with C5, C7, C8 or C11	None or Respond to SRQ, Send IF Msg., pass, Receive control
Electrical Interface	E1 or E2	Open Collector or Tristate

Table 3-2 Required and optional data formats specified by IEEE-488.2.

LISTENER FORMATS	STATUS
<Decimal Numeric Program Data>	Required
<Character Program Data>	Optional
<Suffix Program Data>	Optional
<Non-Decimal Numeric Program Data>	Optional
<String Program Data>	Optional
<Arbitrary Block Program Data>	Optional
<Expression Program Data>	Optional
TALKER FORMATS	
<NR1 Numeric Response Data>	Required
<Arbitrary ASCII Response Data>	Required
<Character Response Data>	Optional
<NR2 Numeric Response Data>	Optional
<NR3 Numeric Response Data>	Optional
<Hexadecimal Numeric Response Data>	Optional
<Octal Numeric Response Data> Optional	
<Binary Numeric Response Data>	Optional
<String Response Data>	Optional
<Definite Length Arbitrary Block Response Data>	Optional
<Indefinite Length Arbitrary Block Response Data>	Optional

formats, some of which are optional for the equipment designer, is illustrated in Table 3-2.

In order to allow older GPIB devices to properly communicate with the newer instruments which are designed to comply with IEEE-488.2, a new concept has been introduced. This is referred to as precise talking and forgiving listening. This requires GPIB devices to accept a wide variety of data formats and codings so that the information transmitted on the bus is received and accepted without error (forgiving listening).

The exact opposite is true when the newer devices must talk on the bus (precise talking). The data transmitted by any IEEE- 488.2 instrument must adhere to a rigorous set of formats. This ensures that new devices will be able to communicate with those using the older formats such as IEEE 728. The concept of precise talking and forgiving listening is very important, and it ensures compatibility between older and newer GPIB devices.

DEVICE MESSAGE PROTOCOLS

Under the original IEEE-488 document, any device which received a message containing more than one command or data would be allowed to interpret and react to that transmission in any way that the manufacturer desired. This freedom of message protocol made it difficult for the software engineer to accurately predict how instruments or devices from various manufacturers would react to any given multiple message.

The IEEE-488.2 document carefully describes a message exchange protocol which is to be used on the IEEE-488.1 bus. In addition to specifying how multiple commands are to be received, it also describes what to do in the event that an incomplete command is sent or a device is interrupted during the processing time of a received message.

A set of device operational states has been defined to implement the device message protocol of IEEE-488.2. These states are tabulated in Table 3-3. Although there is very little chance for confusion when a device receives a single byte from the bus, it may become quite complicated when an incomplete or unterminated command is sent or the device is interrupted. IEEE-488.2 defines what the instrument or device must do under these circumstances.

Included in the device message protocol are specifications on the order in which data bytes are sent and how devices on the bus exchange data. A device cannot send data unless commanded to do so. If a new command is received, the output queue is cleared and that command is processed.

Table 3-3 Definitions of device operational states.

STATE	PURPOSE
IDLE	Wait for messages
READ	Read and execute messages
QUERY	Store responses to be sent
SEND	Send responses
RESPONSE	Complete sending response
DONE	Finished sending response
DEADLOCK	The device cannot buffer more data
UNTERMINATED	The device has attempted to read an unterminated message
INTERRUPTED	The device was interrupted by a new message while sending a response

COMMON COMMAND SET

Since most instruments and devices used in an ATE system use similar commands which perform identical functions, the IEEE-488.2 document has specified a common set of commands which all devices must use. This avoids the problem encountered with IEEE- 488.1 in which devices from various manufacturers used a different set of commands to enable functions and report status. The following set of command groups ensures that all devices communicate uniformly:

1. System data. These commands are used to store or retrieve information such as device identification, descriptions and options, protected user data, and resource description transfer. It is possible to determine the manufacturer and model of the device under remote control. Serial number and device options may also be reported, but this capability is not mandatory. Protected user data and resource description transfer are protected information which can be accessed by the controller only when the protection mechanism, such as a hidden switch, is disabled.

2. Internal operations. These commands include such instrument operations as resetting, self-calibrating, and self-test of a GPIB device. An optional command enables the controller to read the internal settings of the device. The self-calibration feature of the unit must not require any local user interaction and must not cause any conditions that would violate the IEEE-488.1 or 488.2 standards. The device may respond to a calibration query to indicate that the calibration was carried out successfully and report any calibration errors that may have occurred. The Reset command sets the device-dependent functions to a known state and must not affect the state of the IEEE- 488.1 interface, the Service Request Enable register, or Standard Event Status Enable register.
3. Status and event. These commands control the status structure of the GPIB device and provide a means to read and enable events. Included in these commands are Clear, Event Status Enable, Power-on Status, and Service Request Enable. The Clear Status command clears the status register and associated status data structures. The Power-on Status Clear command controls the automatic clearing of the Service Request Enable register, the Standard Event Status Enable register, and the Parallel Poll Enable register. With these registers cleared at power-on, the device is prevented from requesting service while the power-on-clear flag is true. The Service Request Enable command sets the Service Request Enable register which determines what bits in the Status byte will cause a Service Request from the device. The Status Byte Query command reads the status byte, causing the device to respond with an integer in the range of 0 to 255. The binary equivalent of the integer represents the contents of the status byte.
4. Synchronization. The operations of all devices within the system are synchronized with these commands. Included is

a Wait to Continue command which forces the device to complete all previous commands and queries. The Operation Complete command tells the device to set bit 0 in the Standard Event Status register when it completes all pending operations.

5. Parallel poll. The response to a parallel poll is controlled by these commands. It is also possible to obtain the same information from any device, without the need for an actual poll, by executing an Individual Status Query command. This permits the user to determine what an individual device would send on a parallel poll command.
6. Device trigger. These commands enable a device to be triggered and specify how it responds to the trigger message. The Define Device Trigger command stores a sequence of commands which the device will follow when the Group Execute Trigger (GET) is received. The Define Device Trigger query allows the user to review the command sequence which the device will follow upon receipt of the GET comand.
7. Controller. The control of the bus may be passed between devices using the Pass Control Back command. This command tells the potential controller what address to pass control back to. The command must be followed by a number, 0 to 30, which is the address of the device that is to become the next controller. It is also possible to pass control to a device which contains both a primary and secondary address. In this case the command is followed by two numbers.
8. Auto-configure. IEEE-488.2 defines an algorithm which permits the user to automatically assign talk and listen addresses to devices on the bus. Using this optional capability, it is possible to physically assemble a test system and allow the controller to query all devices to find out who they are, and then assign addresses. The Accept Address command allows the controller to assign an address to each configurable device on the bus. As part of the automatic system configuration sequence, the Disable Listener Function command is used to cause a device to stop listening on the bus until it receives a Device Clear (DCL) command.

9. Macros. These optional commands enable the user to define new commands for the instrument under control. Macros can be used to provide shorthand for complex commands and reduce bus traffic. Other instruments can be emulated using these commands. The Define Macro command is used to assign a sequence of commands to a macro label. When the device receives the macro label as a command, it executes the sequence of commands contained within the macro. The macro label cannot be the same as a common command or query but may be the same as a device-dependent command. The Enable Macro command, followed by number 0, disables all macros so that a device dependent command which is the same as a macro can be executed. If the Enable Macro command is followed by a number in the range of -32767 to 32767, the macros will be enabled. The user can determine if macros are enabled on a device by sending the Enable Macro query. The Learn Macro query causes the device to respond with the labels of all defined macros, whether or not they are enabled. Macros may be purged from a device by using the Purge Macros command.
10. Stored Settings. These commands are used to save the state of the device under control, to be used at a later time. The Save command stores the present state of the device in the device's memory. If there is more than one location in which this data can be stored, the command is followed by a number which designates the storage register to use. The Recall command restores the state of the device, as stored in its memory from a previous Save command. As with the Save command, the Recall command must be followed by

a number to specify the register from which the stored settings are to be selected.

Note that all common commands are always sent in the data mode of the bus (ATN false). IEEE-488.2 specifies that certain common commands must be operational in a GPIB device, while others are left up to the discretion of the instrument designer. Table 3-4 illustrates the sets of the common commands, organized by command group. A description of the common commands is in Appendix E.

STATUS REPORTING

Status reporting defined by IEEE-488.2 builds upon and extends the original specifications of the status byte of the 488.1 document. Figure 3-2 illustrates the IEEE-488.2 status reporting model showing the IEEE-488.1 status byte, which can be read by either a serial poll or Status Byte Query. The 488.2 byte contains seven single-bit summary messages from Status Data Structures which are registers or queues. IEEE-488.1 defines the Recall Status Query (RSQ) bit, and IEEE-488.2 defines the event status bit (ESB) and message available bit (MAV). The user can enable a GPIB device to request service, depending upon the state of the summary bits of the status byte.

The status byte is transferred to the controller by means of the IEEE-488.1 serial poll or an IEEE-488.2-defined common query. Additionally, more common commands and queries are defined to obtain information from the devices under remote control. An overview of the IEEE-488.2 status reporting structure is shown in Figure 3-2.

The Status Byte register, illustrated in Figure 3-3, was originally defined by IEEE-488.1, which did not specify how the bits (other

Table 3-4 IEEE-488.2 comman commands, organized by command group.

MNEMONIC	DISCRIPTION	COMPLIANCE
	AUTO CONFIGURE COMMANDS	
• AAD	Assign Address	Opt.
• DLF	Disable Listener Function	Opt.
	SYSTEM DATA COMMANDS	
• IDN?	Identification Query	Reqd.
• OPT?	Option Identification Query	Opt.
• PUD	Protected User Data	Opt.
• PUD?	Protected User Data Query	Opt.
• RDT	Resource Description Transfer	Opt.
• RDT?	Resource Description Transfer Query	Opt.
	INTERNAL OPERATION COMMANDS	
• CAL	Calibration Query	Opt.
• LRN	Learn Device Setup Query	Opt.
• RST	Reset	Reqd.
• TST?	Self-Test Query	Reqd.
	SYNCHRONIZATION COMMANDS	
• OPC	Operation Complete	Reqd.
• OPC?	Operation Complete Query	Reqd.
• WAI	Wait to Complete	Reqd.
	MACRO COMMANDS	
• DMC	Define Macro	Opt.
• EMC	Enable Macro	Opt.
• EMC?	Enable Macro Query	Opt.
• GMC?	Get Macro Contents Query	Opt.
• LMC?	Learn Macro Query	Opt.
• PMC	Purge Macros	Opt.
	PARALLEL POLL COMMANDS	
• IST?	Individual Status Query	Reqd. If PP1
• PRE	Parallel Poll Enable Register Enable	Reqd. if PP1
• PRE?	Parallel Poll Enable Reg Enable Query	Reqd. If PP1
	STATUS & EVENT COMMANDS	
• CLS	Clear Status	Reqd.
• ESE	Event Status Enable	Reqd.
• ESE?	Event Status Enable Query	Reqd.
• ESR?	Event Status Register Query	Reqd.
• PSC	Power on Status Clear	Opt.
• PSC?	Power on Status Clear Query	Opt.
• SRE	Service Request Enable	Reqd.
• SRE?	Service Request Enable Query	Reqd.
• STB?	Read Status Byte Query	Reqd.
	DEVICE TRIGGER COMMANDS	
• DDT	Define Device Trigger	Opt. if DT1
• DDT?	Define Device Trigger Query	Opt. if DT1
• TRG	Trigger	Reqd. if DT1
	CONTROLLER COMMANDS	
• PCB	Pass Control Back	Reqd. If Controller
	STORED SETTINGS COMMANDS	
• RCL	Recall Instrument State	Opt.
• SAV	Save Instrument State	Opt.

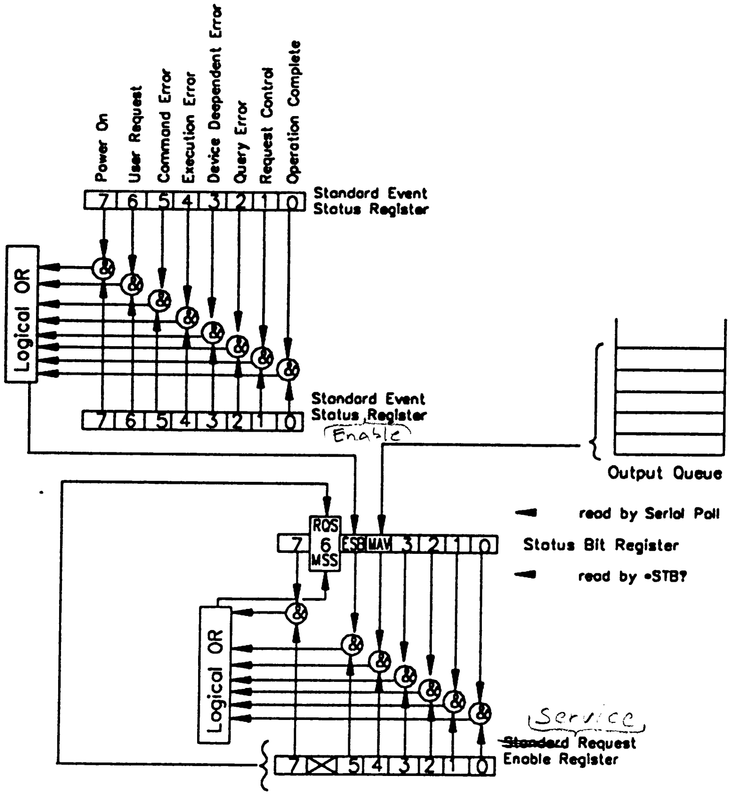


Figure 3-2 IEEE-488.2 status reporting model.

than RSQ) were to be set or cleared. This was left up to the discretion of the equipment manufacturer.

IEEE-488.2 defines additional commands which allow the user to access the status byte and associated data structures. Although a serial poll will clear the RSQ bit, it will not clear the status byte, which can be done by clearing the related status structures using the *CLS command.

The bits defined by IEEE-488.2 are bits 4, 5, and 6. Bit 4 is the message available bit (MAV) which is true if the output queue

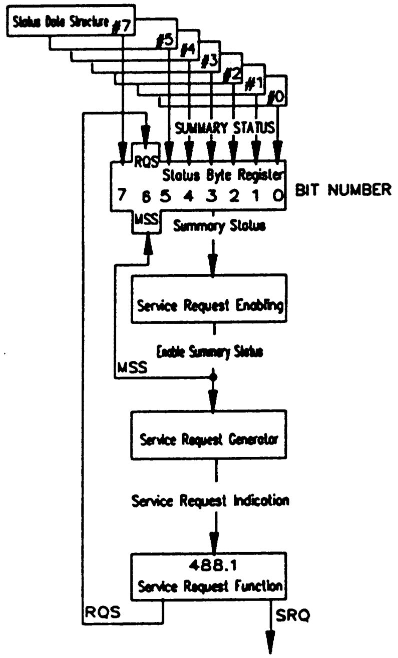


Figure 3-3 IEEE-488.2 status reporting structure.

contains data available to output. Bit 5, event status bit (ESB), indicates if an enabled standard event has occurred. The master summary status bit (MSS) is bit 6, which indicates if the device has at least one condition to request service. Note that the MSS bit is not considered part of the IEEE-488.1 status byte and will not be sent in response to a serial poll. The RSQ bit, however, if set, will be sent in a 488.1 serial poll.

Figure 3-4 illustrates the service request enabling operation. The user may set bits in the Service Request Enable register (SRER), corresponding to bits in the status byte. When a bit is set in the SRER, it enables that bit in the status register to request service.

Event registers are used to remember that a predefined condition changes in a device. IEEE-488.2 defines a command to read the Standard Event Status registor, but if a unit has more than one event

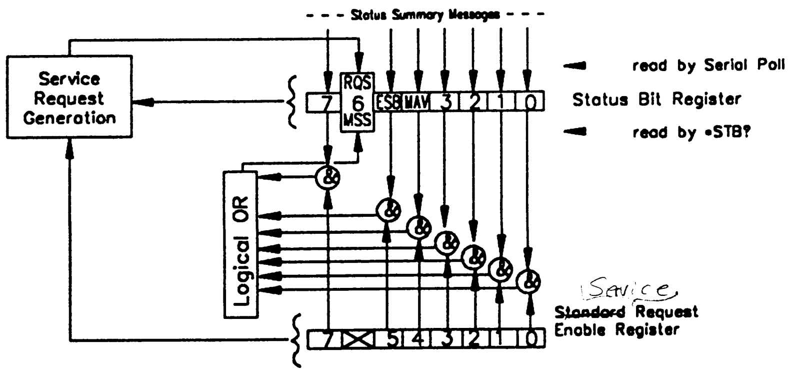


Figure 3-4 Service request enabling operation.

register, there must be other device-dependent commands to access the data stored within it. The bits, once set, cannot be cleared until done so by either reading the register or using the Clear command (*CLS). IEEE-488.2 defines the transition criteria which sets an event bit true. This occurs when its condition changes from either false to true or true to false.

A GPIB device may also provide Event Enable registers, which are similar to the Service Request Enable register. Setting bits in the Event Enable register permits bits in the Event register to be summarized in the status byte.

The Standard Event Status register (SESR) is a specific application of status reporting, and the IEEE-488.2 document specifies the meaning of each bit of the SESR. Figure 3-5 depicts the Standard Event Status register. The 8 bits of the SESR have been defined by IEEE-488.2 as specific conditions which can be monitored and reported back to the user upon request. These events are:

Bit 0, operation complete (OPC). This bit, generated in response to the OPC command, is set when the device has completed its current operations and is ready to accept a new command.

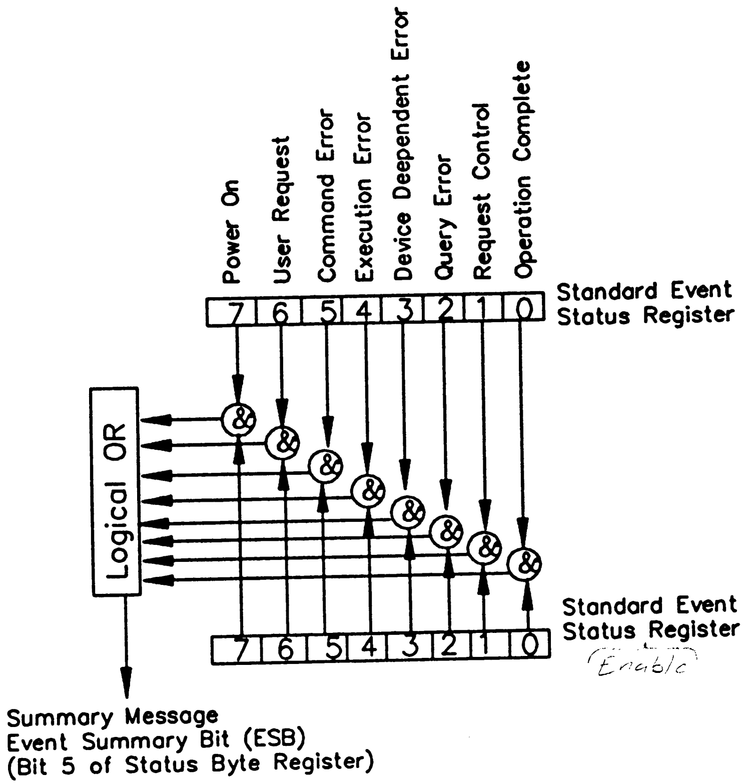


Figure 3-5 Standard event status register.

Bit 1, request control (RQC). This bit is used by a device to indicate to the controller that it wants to become the active controller on the bus.

Bit 2, query error (QYE). A query error, indicated by this bit, occurs when an attempt to read data from the output queue is made when no data is present or if some data was lost as in the event of a queue overflow.

Bit 3, device-dependent error (DDE). This bit is set when an error in a device function occurs, such as improper execution of a command due to an internal condition or malfunction of the device.

Bit 4, execution error (EXE). An execution error occurs when the command received by the device is not within its legal capability, or is not consistent with its designed operation. It may also occur when an internal condition prevents proper execution of a valid command.

Bit 5, command error (CME). This bit indicates that the device received a command that was a syntax error (not defined by 488.2 standard), a semantic error such as a misspelled command, or a command that the device does not implement. A group execute trigger (GET) received inside a program message will also cause a command error.

Bit 6, user request (URQ). This bit, set without regard to the remote or local state of a device, will be set when the user has activated a device defined control. Its purpose is to enable the user to obtain the controller's attention.

Bit 7, power on (PON). This bit indicates that the device's power source was turned off, then on, since the last time that the SSER was read.

All IEEE-488.2 devices have the Standard Event Status register and may also contain other event registers. The SESR is written with the Enable Status (*ESE) command and read with the Enable Status query (*ESE?). The register is automatically cleared when reading it via the ESE? query or sending the Clear command (CLS). The device designer has the option of clearing the register with the power-on transition and recording any transitions which occur subsequently.

Queues are used to allow a device to report to the controller status or other information in an orderly manner. Each queue has a summary message bit which is set when the queue contains information. The output queue of the IEEE-488.2 device uses the MAV bit in the status byte to indicate that it contains available data. This is a first-in, first-out (FIFO) queue, and it can be cleared only by a 488.1 Device Clear command, the Reset command, or by power on.

PARALLEL POLL

The parallel poll specified by the IEEE-488.1 document provided a means to quickly ascertain the status of each of the devices on the bus. IEEE-488.2 carries this capability further with an optional means of generating and controlling a device's response to a parallel poll. Figure 3-6 illustrates the structure of the 488.2 parallel poll data handling response. The structure of the parallel poll is the same as the Event register, but its summary bit is sent in response to a parallel poll and not in a status byte. This summary bit is referred to as IST, or individual status local message. An Enable register is provided to determine which events are summarized in the ist.

The Individual Status Query (*IST?) allows the user to determine the current state of the IST local message. This is the status that would be sent in response to a parallel poll. The query permits reading the response of the device without the need to perform an actual parallel poll.

The Parallel Poll Enable Register command (*PRE) sets the bits which determine what conditions are summarized in the IST. The command must be followed by a number which, when converted to binary, represents the bits set in the Parallel Poll Enable register.

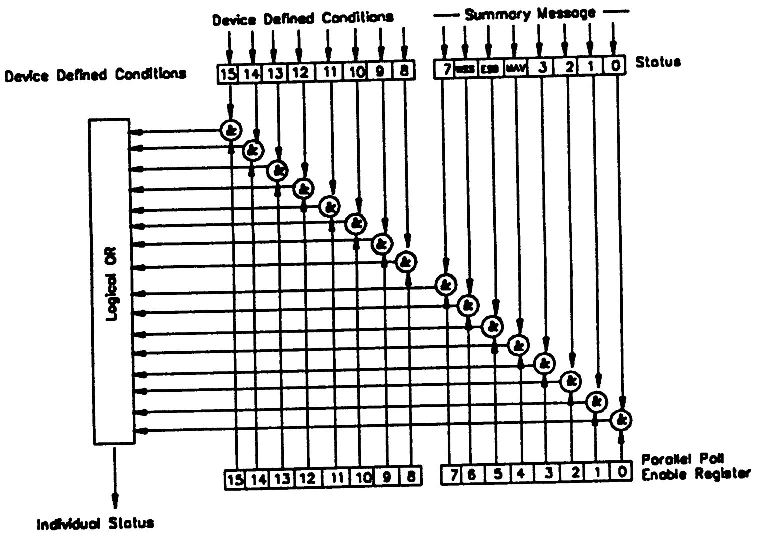


Figure 3-6 IEEE-488.2 parallel poll data handling response.

The Parallel Poll Enable Register Query (*PRE?) allows the device to respond with an integer in the range of 0 to 65535. This tells the user what bits are set in the PPER.

type of data on 24 lines. Model 4874B is implemented for interface functions SH1, AH1, TE5, LE3, SR1, PP1, RL0, DC1, DT1, E1, and C0.

INTERFACE FUNCTIONS AND SUBSETS

Most manufacturers of instruments and devices which are GPIB controllable list the various interface functions which are implemented in the product. These capabilities can usually be found in the data sheets for the product, and in some cases they are marked on the instrument itself (usually near the GPIB connector). Although there are just 11 basic interface functions, the various subsets of these functions number much more than this. Table 5-1 illustrates the various subsets of each of the basic interface functions.

In Table 5-1 under the talker and listener functions, extended talker and extended listener refer to those devices which are capable of accepting secondary address commands. Additionally, MTA and MLA are the mnemonic for "my talk address" and "my listen address." When an addressed device receives the command to talk or listen, the command is referred to as MTA or MLA.

Although only six subsets of the controller function are listed, it should be noted that the IEEE-488 document specifies a total of 29 controller subsets. Those tabulated in Table 5-1 are the most significant levels.

When selecting instruments and devices to be used in a GPIB system, it is important to note the interface function subsets of which any product of interest is capable. These are specified by the manufacturer of the device who has determined what functions (and at what level) are required for the product to perform its desired capabilities. It is entirely possible that one or more desired interface functions are completely absent in a unit under consideration. Should this occur, it will be necessary to seek out a different product

Table 5-1. Table of interface functions and subsets.

Interface Function	Basic Code	Capability Code
Source Handshake	SH	SH0 — No capability SH1 — Full capability
Acceptor Handshake	AH	AH0 — No capability AH1 — Full capability
Talker (extended talker)	T(TE)	T(TE)0 — No capability T(TE)1 — Basic talker, serial poll, talk only T(TE)2 — Basic talker, serial poll T(TE)3 — Basic talker, talk only T(TE)4 — Basic talker T(TE)5 — Basic talker, serial poll, talk only, unaddresses if MLA T(TE)6 — Basic talker, serial poll, unaddresses if MLA T(TE)7 — Basic talker, talk only, unaddresses if MLA T(TE)8 — Basic talker, unaddresses if MLA
Listener (extended listener)	L(LE)	L(LE)0 — No capability L(LE)1 — Basic listener, listen only L(LE)2 — Basic listener L(LE)3 — Basic listener, Listen only, unaddresses if MTA L(LE)4 — Basic listener, unaddresses if MTA
Service Request	SR	SR0 — No capability SR1 — Full capability
Remote/Local	RL	RL0 — No capability RL1 — Full capability RL2 — No local lockout
Parallel poll	PP	PP0 — No capability PP1 — Remote configuration PP2 — Local configuration
Device clear	DC	DC0 — No capability DC1 — Full capability DC2 — Omitselective device clear
Device trigger	DT	DT0 — No capability DT1 — Full capability
Driver electronics	E	E1 — Open collector, 250 Kilobytes/sec max E2 — Tri-state, 1 Megabyte/sec max
Controller	C	C0 — No capability C1 — System controller C2 — Send IFC and take charge C3 — Send REN C4 — Respond to service request C5 — Send interface messages, receive, control pass control to self, parallel poll, take control synchronously

Table 5-1. *(continued)*

or work around the deficiency through the use of programming techniques (if possible).

One should be aware that it is possible that one instrument in a GPIB system may not be compatible with others if they each do not share the same desired interface function. For example, if it is necessary for local control of all instruments in a system to be locked out to prevent tampering by personnel, but the power supply feeding the unit under test does not have the local lockout function, the test engineer would not be able to implement this feature to cover all instruments in the ATE system.

PROGRAMMING REQUIREMENTS

When a GPIB interface system is fully assembled, complete with controller, it is virtually useless until the program or software is written and loaded into the controller's memory. The program will contain the instructions which will cause each and every device on the bus to perform the desired task.

ASCII och GPIB-koder

B7 B6 B5	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
BITS	CONTROL		NUMBERS SYMBOLS		UPPER CASE		LOWER CASE	
B4 B3 B2 B1								
Ø Ø Ø Ø	Ø NUL Ø Ø	2Ø DLE 1Ø 16	4Ø SP 2Ø 32	6Ø 0 3Ø 48	1ØØ @ 4Ø 64	12Ø P 5Ø 8Ø	14Ø ` 6Ø 96	16Ø p 7Ø 112
Ø Ø Ø 1	1 GTL SOH 1 1	21 LLO DC1 11 17	41 ! 21 33	61 1 31 49	1Ø1 A 41 65	121 Q 51 81	141 a 61 97	161 q 71 113
Ø Ø 1 Ø	2 STX 2 2	22 DC2 12 18	42 " 22 34	62 2 32 5Ø	1Ø2 B 42 66	122 R 52 82	142 b 62 98	162 r 72 114
Ø Ø 1 1	3 ETX 3 3	23 DC3 13 19	43 # 23 35	63 3 33 51	1Ø3 C 43 67	123 S 53 83	143 c 63 99	163 s 73 115
Ø 1 Ø Ø	4 SDC EOT 4 4	24 DCL DC4 14 2Ø	44 $ 24 36	64 4 34 52	1Ø4 D 44 68	124 T 54 84	144 d 64 1ØØ	164 t 74 116
Ø 1 Ø 1	5 PPC ENQ 5 5	25 PPU NAK 15 21	45 % 25 37	65 5 35 53	1Ø5 E 45 69	125 U 55 85	145 e 65 1Ø1	165 u 75 117
Ø 1 1 Ø	6 ACK 6 6	26 SYN 16 22	46 & 26 38	66 6 36 54	1Ø6 F 46 7Ø	126 V 56 86	146 f 66 1Ø2	166 v 76 118
Ø 1 1 1	7 BEL 7 7	27 ETB 17 23	47 ' 27 39	67 7 37 55	1Ø7 G 47 71	127 W 57 87	147 g 67 1Ø3	167 w 77 119
1 Ø Ø Ø	1Ø GET BS 8 8	3Ø SPE CAN 18 24	5Ø (28 4Ø	7Ø 8 38 56	11Ø H 48 72	13Ø X 58 88	15Ø h 68 1Ø4	17Ø x 78 12Ø
1 Ø Ø 1	11 TCT HT 9 9	31 SPD EM 19 25	51) 29 41	71 9 39 57	111 I 49 73	131 Y 59 89	151 i 69 1Ø5	171 y 79 121
1 Ø 1 Ø	12 LF A 1Ø	32 SUB 1A 26	52 * 2A 42	72 : 3A 58	112 J 4A 74	132 Z 5A 9Ø	152 j 6A 1Ø6	172 z 7A 122
1 Ø 1 1	13 VT B 11	33 ESC 1B 27	53 + 2B 43	73 ; 3B 59	113 K 4B 75	133 [5B 91	153 k 6B 1Ø7	173 { 7B 123
1 1 Ø Ø	14 FF C 12	34 FS 1C 28	54 , 2C 44	74 < 3C 6Ø	114 L 4C 76	134 \ 5C 92	154 l 6C 1Ø8	174 | 7C 124
1 1 Ø 1	15 CR D 13	35 GS 1D 29	55 - 2D 45	75 = 3D 61	115 M 4D 77	135] 5D 93	155 m 6D 1Ø9	175 } 7D 125
1 1 1 Ø	16 SO E 14	36 RS 1E 3Ø	56 . 2E 46	76 > 3E 62	116 N 4E 78	136 ^ 5E 94	156 n 6E 11Ø	176 ~ 7E 126
1 1 1 1	17 SI F 15	37 US 1F 31	57 / 2F 47	77 ? 3F 63	117 O 4F Ø	137 _ 5F 95	157 o 6F 111	177 RUBOUT (DEL) 7F 127
	ADDRESSED COMMANDS	UNIVERSAL COMMANDS	LISTEN ADDRESSES		TALK ADDRESSES		SECONDARY ADDRESSES OR COMMANDS	

KEY:

octal	25	PPU	Message Mnemonic
	NAK		ASCII/ISO character
hex	15	21	decimal

Appendix B ASCII and IEEE488 codes

The following table lists all the ASCII codes and the corresponding IEEE-488 meaning of each.

Decimal	Hex.	ASCII	IEEE488
0	00	NULL	
1	01	SOH	GTL
2	02	STX	
3	03	ETX	
4	04	EOT	SDC
5	05	ENQ	PPC
6	06	ACK	
7	07	BELL	
8	08	BS	GET
9	09	HT	TCT
10	0A	LF	
11	0B	VT	
12	0C	FF	
13	0D	CR	
14	0E	SO	
15	0F	SI	
16	10	DLE	
17	11	DC1	LLO
18	12	DC2	
19	13	DC3	
20	14	DC4	DCL
21	15	NACK	PPU
22	16	SYNC	
23	17	ETB	
24	18	CAN	SPE
25	19	EM	SPD
26	1A	SUB	
27	1B	ESC	
28	1C	FS	
29	1D	GS	
30	1E	RS	
31	1F	US	
32	20	space	LA0
33	21	!	LA1
34	22	“	LA2
35	23	#	LA3
36	24	$	LA4
37	25	%	LA5
38	26	&	LA6
39	27	‘	LA7
40	28	(	LA8
41	29	)	LA9
42	2A	*	LA10
43	2B	+	LA11
44	2C	,	LA12
45	2D	–	LA13
46	2E	.	LA14
47	2F	/	LA15
48	30	0	LA16
49	31	1	LA17
50	32	2	LA18
51	33	3	LA19
52	34	4	LA20
53	35	5	LA21
54	36	6	LA22
55	37	7	LA23
56	38	8	LA24
57	39	9	LA25
58	3A	:	LA26
59	3B	;	LA27
60	3C	<	LA28
61	3D	=	LA29
62	3E	>	LA30

LA = Listen Address

Decimal	Hex.	ASCII	IEEE488
63	3F	?	UNL
64	40	@	TA0
65	41	A	TA1
66	42	B	TA2
67	43	C	TA3
68	44	D	TA4
69	45	E	TA5
70	46	F	TA6
71	47	G	TA7
72	48	H	TA8
73	49	I	TA9
74	4A	J	TA10
75	4B	K	TA11
76	4C	L	TA12
77	4D	M	TA13
78	4E	N	TA14
79	4F	O	TA15
80	50	P	TA16
81	51	Q	TA17
82	52	R	TA18
83	53	S	TA19
84	54	T	TA20
85	55	U	TA21
86	56	V	TA22
87	57	W	TA23
88	58	X	TA24
89	59	Y	TA25
90	5A	Z	TA26
91	5B	[	TA27
92	5C	\	TA28
93	5D	]	TA29
94	5E	^	TA30
95	5F	–	UNT
96	60	`	SC0
97	61	a	SC1
98	62	b	SC2
99	63	c	SC3
100	64	d	SC4
101	65	e	SC5
102	66	f	SC6
103	67	g	SC7
104	68	h	SC8
105	69	i	SC9
106	6A	j	SC10
107	6B	k	SC11
108	6C	l	SC12
109	6D	m	SC13
110	6E	n	SC14
111	6F	o	SC15
112	70	p	SC16
113	71	q	SC17
114	72	r	SC18
115	73	s	SC19
116	74	t	SC20
117	75	u	SC21
118	76	v	SC22
119	77	w	SC23
120	78	x	SC24
121	79	y	SC25
122	7A	z	SC26
123	7B	{	SC27
124	7C	|	SC28
125	7D	}	SC29
126	7E	~	SC30
127	7F	DEL	SC31

TA= TalkAddress

SC = Secondary Command

Ur National Instruments: Instrumentation reference and catalog '97

GPIB Tutorial

In 1965, Hewlett-Packard designed the Hewlett-Packard Interface Bus (HP-IB) to connect their line of programmable instruments to their computers. Because of its high transfer rates (nominally 1 Mbytes/s), this interface bus quickly gained popularity. It was later accepted as IEEE Standard 488-1975, and has evolved to ANSI/IEEE Standard 488.1-1987. Today, the name General Purpose Interface Bus (GPIB) is more widely used than HP-IB. ANSI/IEEE 488.2-1987 strengthened the original standard by defining precisely how controllers and instruments communicate. Standard Commands for Programmable Instruments (SCPI) took the command structures defined in IEEE 488.2 and created a single, comprehensive programming command set that is used with any SCPI instrument. Figure 1 summarizes GPIB history.

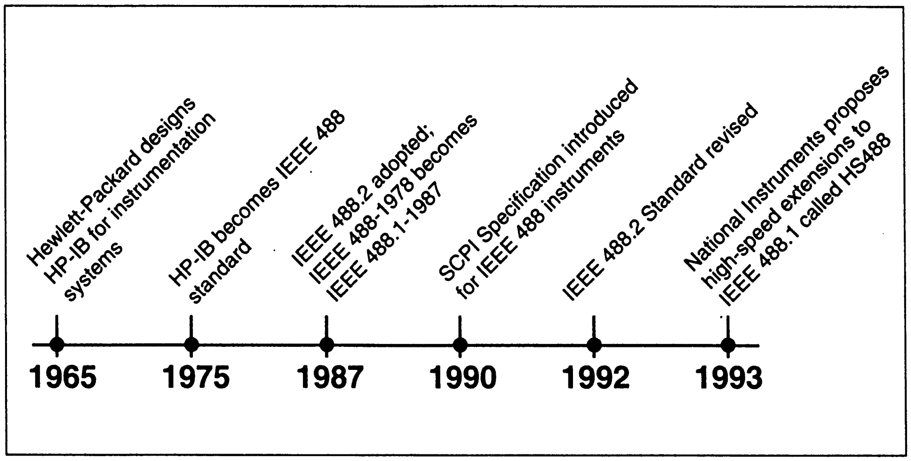

Hewlett-Packard designs HP-IB for instrumentation systems
HP-IB becomes IEEE 488 standard
IEEE 488.2 adopted; IEEE 488-1978 becomes IEEE 488.1-1987
SCPI Specification introduced for IEEE 488 instruments
IEEE 488.2 Standard revised
National Instruments proposes high-speed extensions to IEEE 488.1 called HS488
1965
1975
1987
1990
1992
1993

Figure 1. GPIB History

Types of GPIB Messages

GPIB devices communicate with other GPIB devices by sending device-dependent messages and interface messages through the interface system.

- Device-dependent messages, often called data or data messages, contain device-specific information, such as programming instructions, measurement results, machine status, and data files.
- Interface messages manage the bus. Usually called commands or command messages, interface messages perform such functions as initializing the bus, addressing and unaddressing devices, and setting device modes for remote or local programming.

The term "command" as used here should not be confused with some device instructions that are also called commands. Such device-specific commands are actually data messages as far as the GPIB interface system itself is concerned.

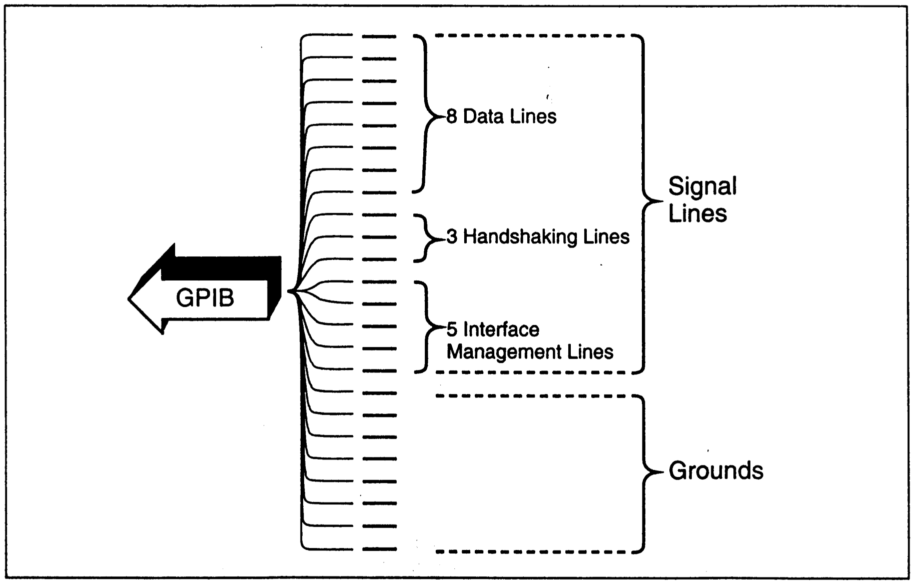


Figure 2. GPIB Signals and Lines

Talkers, Listeners, and Controllers

GPIB Devices can be Talkers, Listeners, and/or Controllers. A Talker sends data messages to one or more Listeners, which receive the data. The Controller manages the flow of information on the GPIB by sending commands to all devices. A digital voltmeter, for example, is a Talker and is also a Listener.

The GPIB is like an ordinary computer bus, except that a computer has its circuit cards interconnected via a backplane – the GPIB has stand-alone devices interconnected by standard cables.

The role of the GPIB Controller is comparable to the role of a computer CPU, but a better analogy is to compare the Controller to the switching center of a city telephone system.

The switching center (Controller) monitors the communications network (GPIB). When the center (Controller) notices that a party (device) wants to make a call (send a data message), it connects the caller (Talker) to the receiver (Listener).

The Controller usually addresses (or enables) a Talker and a Listener before the Talker can send its message to the Listener. After the message is transmitted, the Controller may address other Talkers and Listeners.

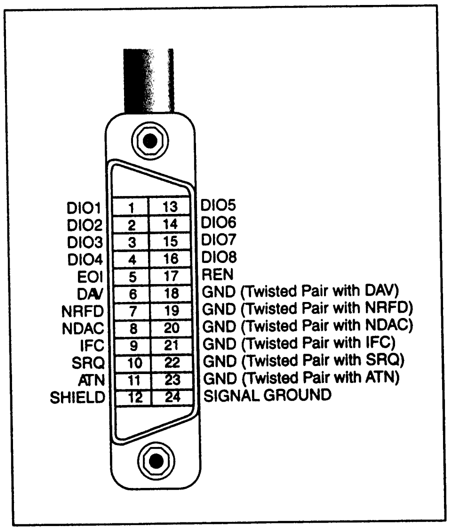

DIO1
DIO2
DIO3
DIO4
EOI
DAV
NRFD
NDAC
IFC
SRQ
ATN
SHIELD
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
DIO5
DIO6
DIO7
DIO8
REN
GND (Twisted Pair with DAV)
GND (Twisted Pair with NRFD)
GND (Twisted Pair with NDAC)
GND (Twisted Pair with IFC)
GND (Twisted Pair with SRQ)
GND (Twisted Pair with ATN)
SIGNAL GROUND

Figure 3. GPIB Connector and Signal Assignment

Some GPIB configurations do not require a Controller. For example, a device that is always a Talker, called a talk-only device, is connected to one or more listen-only devices.

A Controller is necessary when the active or addressed Talker or Listener must be changed. The Controller function is usually handled by a computer.

A computer equipped with National Instruments GPIB hardware and software can perform the roles of Talker/Listener and Controller.

The Controller-In-Charge and System Controller

Although there can be multiple Controllers on the GPIB, at any time only one Controller is the Controller-In-Charge (CIC). Active control can be passed from the current CIC to an idle Controller. Only the System Controller can make itself the CIC. The National Instruments GPIB board is usually the System Controller.

GPIB Signals and Lines

The GPIB interface system consists of 16 signal lines and eight ground-return or shield-drain lines. The 16 signal lines, discussed below, are grouped into data lines (eight), handshake lines (three), and interface management lines (five) (see Figure 2).

Data Lines

The eight data lines, DIO1 through DIO8, carry both data and command messages. The state of the Attention (ATN) line determines whether the information is data or commands. All commands and most data use the 7-bit ASCII or ISO code set, in which case the eighth bit, DIO8, is either unused or used for parity.

Handshake Lines

Three lines asynchronously control the transfer of message bytes between devices. The process is called a 3-wire interlocked handshake. It guarantees that message bytes on the data lines are sent and received without transmission error.

- NRFD (not ready for data) – Indicates when a device is ready or not ready to receive a message byte. The line is driven by all devices when receiving commands, by Listeners when receiving data messages, and by the Talker when enabling the HS488 protocol.

- NDAC (not data accepted) – Indicates when a device has or has not accepted a message byte. The line is driven by all devices when receiving commands, and by Listeners when receiving data messages.

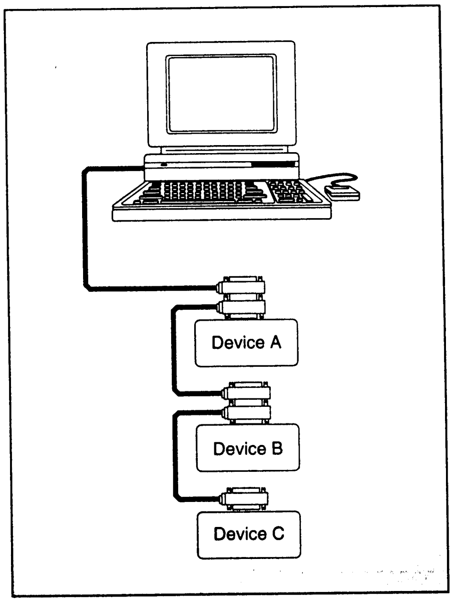


Figure 4. Linear Configuration

- DAV (data valid) – Tells when the signals on the data lines are stable (valid) and can be accepted safely by devices. The Controller drives DAV when sending commands, and the Talker drives DAV when sending data messages.

Interface Management Lines

Five lines manage the flow of information across the interface.

- ATN (attention) – The Controller drives ATN true when it uses the data lines to send commands, and drives ATN false when a Talker can send data messages.
- IFC (interface clear) – The System Controller drives the IFC line to initialize the bus and become CIC.
- REN (remote enable) – The System Controller drives the REN line, which is used to place devices in remote or local program mode.
- SRQ (service request) – Any device can drive the SRQ line to asynchronously request service from the Controller.
- EOI (end or identify) – The EOI line has two purposes – The Talker uses the EOI line to mark the end of a message string, and the Controller uses the EOI line to tell devices to identify their response in a parallel poll.

Physical and Electrical Characteristics

Devices are usually connected with a shielded 24-conductor cable with both a plug and receptacle connector at each end (see Figure 3). You can link devices in either a linear configuration (see Figure 4), a star configuration (see Figure 5, page 4-13), or a combination of the two.

The standard connector is the Amphenol or Cinch Series 57 MICRORIBBON or AMP CHAMP type. For special interconnect applications, an adapter cable with non-standard cable and/or connectors is used.

The GPIB uses negative logic with standard TTL levels. When DAV is true, for example, it is a TTL low level (≤ 0.8 V), and when DAV is false, it is a TTL high level (≥ 2.0 V).

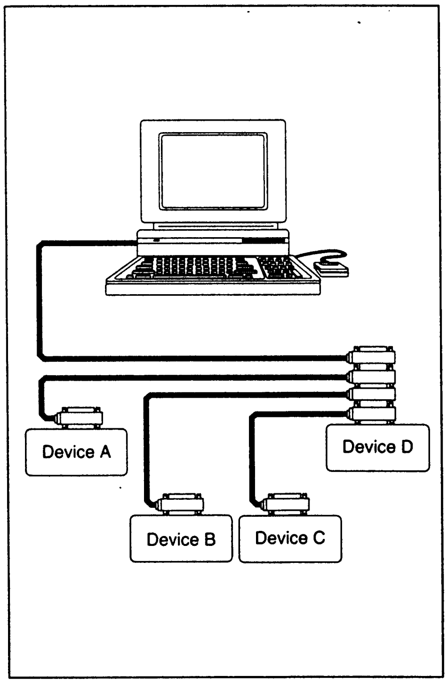


Figure 5. Star Configuration

Configuration Requirements

To achieve the high data transfer rate for which the GPIB was designed, the physical distance between devices and the number of devices on the bus are limited.

The following restrictions are typical for normal operation:

- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus
- A maximum total cable length of 20 m
- No more than 15 device loads connected to each bus, with no less than two-thirds powered on

For higher speed systems using the 3-wire IEEE 488.1 handshake (T1 delay = 350 ns), and HS488 systems, the following restrictions apply:

- A maximum total cable length of 15 m with a device load per 1 m cable
- All devices should be powered on
- All devices should use 48 mA three-state drivers
- Device capacitance on each GPIB signal should be less than 50 pF per device

Bus extenders and an expander are available from National Instruments when your system exceeds these limits.

IEEE 488.2 and SCPI

The SCPI and IEEE 488.2 standards addressed the limitations and ambiguities of the original IEEE 488 standard. IEEE 488.2 makes it possible to design more compatible and productive test systems. SCPI simplifies the programming task by defining a single comprehensive command set for programmable instrumentation, regardless of type or manufacturer. The scope of each of the IEEE 488, IEEE 488.2, and SCPI standards is shown in Figure 6. The ANSI/IEEE Standard 488-1975, now called IEEE 488.1, greatly simplified the interconnection of programmable instrumentation by clearly defining mechanical, electrical, and hardware protocol specifications. For the first time, instruments from different manufacturers were interconnected by a standard cable. Although this standard went a long way towards improving the productivity of test engineers, the standard did have a number of shortcomings. Specifically, IEEE 488.1 did not address data formats, status reporting, message exchange protocol, common configuration commands, or device-specific commands. As a result, each manufacturer implemented these items differently, leaving the test system developer with a formidable task.

IEEE 488.2 enhanced and strengthened IEEE 488.1 by standardizing data formats, status reporting, error handling, Controller functionality, and common commands to which all instruments must respond in a defined manner. By standardizing these issues, IEEE 488.2 systems are much more compatible and reliable. The IEEE 488.2 standard focuses mainly on the software protocol issues and thus maintains compatibility with the hardware-oriented IEEE 488.1 standard.

SCPI built on the IEEE 488.2 standard and defined device-specific commands that standardize programming instruments. SCPI systems are much easier to program and maintain. In many cases, you can interchange or upgrade instruments without having to change the test program. The combination of SCPI and IEEE 488.2 offers significant productivity gains, and finally, delivers as sound a software standard as IEEE 488.1 did a hardware standard.

IEEE 488.2

IEEE Standard 488.2-1987 encouraged a new level of growth and acceptance of the

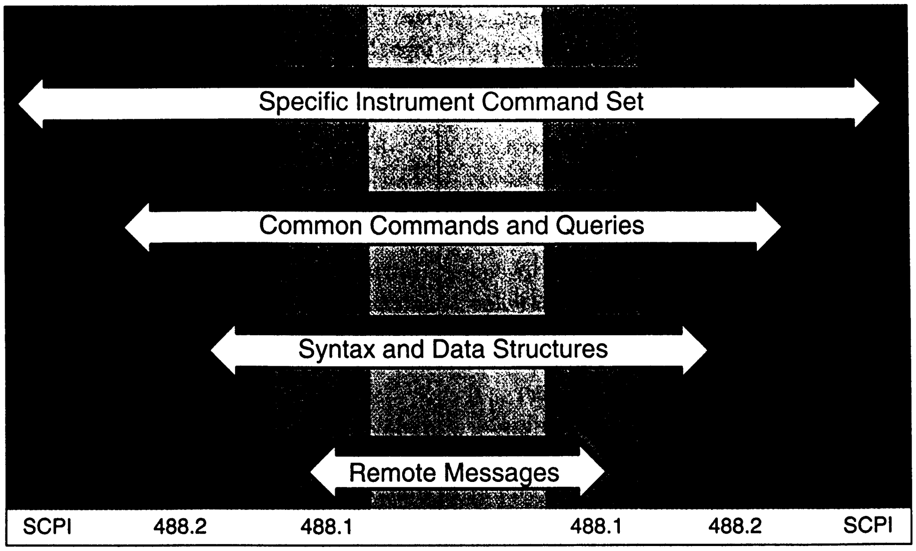

Specific Instrument Command Set
Common Commands and Queries
Syntax and Data Structures
Remote Messages
SCPI
488.2
488.1
488.1
488.2
SCPI

Figure 6. Evolution of GPIB Instrumentation Standards

IEEE 488 bus or GPIB by addressing problems that had arisen from the original IEEE 488 standard. IEEE 488.2 was drafted on the premise that it stay compatible with the existing IEEE 488.1 standard. The overriding concept used in the IEEE 488.2 specification for the communication between Controllers and instruments is that of "precise talking" and "forgiving listening." In other words, IEEE 488.2 exactly defined how both IEEE 488.2 Controllers and IEEE 488.2 instruments talk so that a completely IEEE 488.2-compatible system can be highly reliable and efficient. The standard also required that IEEE 488.2 devices be able to work with existing IEEE 488.1 devices by accepting a wide range of commands and data formats as a Listener. You obtain the true benefits of IEEE 488.2 when you have a completely IEEE 488.2-compatible system.

Controllers

Although IEEE 488.2 had less impact on Controllers than it did on instruments, there are several requirements and optional improvements for Controllers that made an IEEE 488.2 Controller a necessary component of test systems. IEEE 488.2 precisely defined the way IEEE 488.2 Controllers send commands and data and added functionality. Because of these IEEE 488.2 Controller requirements, instrument manufacturers can design more compatible and efficient instruments. The benefits of this standardization for the test system developer are reduced development time and cost, because it solves the problems caused by instrument incompatibilities, varying command structures, and data formats.

Requirements of IEEE 488.2 Controllers – IEEE 488.2 defined a number of requirements for a Controller, including an exact set of IEEE 488.1 interface capabilities, such as pulsing the interface clear line for 100 µs, setting and detecting EOI, setting/asserting the REN line, sensing the state and transition of the SRQ line, sensing the state of NDAC, and timing out on any I/O transaction. Other key requirements for Controllers are bus control sequences and bus protocols.

Description	**Control Sequence**	**Compliance**
Send ATN-true commands	SEND COMMAND	Mandatory
Set address to send data	SEND SETUP	Mandatory
Send ATN-false data	SEND DATA BYTES	Mandatory
Send a program message	SEND	Mandatory
Set address to receive data	RECEIVE SETUP	Mandatory
Receive ATN-false data	RECEIVE RESPONSE MESSAGE	Mandatory
Receive a response message	RECEIVE	Mandatory
Pulse IFC line	SEND IFC	Mandatory
Place devices in DCAS	DEVICE CLEAR	Mandatory
Place devices in local state	ENABLE LOCAL CONTROLS	Mandatory
Place devices in remote state	ENABLE REMOTE	Mandatory
Place devices in remote with local lockout state	SET RWLS	Mandatory
Place devices in local lockout state	SEND LLO	Mandatory
Read IEEE 488.1 status byte	READ STATUS BYTE	Mandatory
Send group execution trigger (GET) message	TRIGGER	Mandatory
Give control to another device	PASS CONTROL	Optional
Conduct a parallel poll	PERFORM PARALLEL POLL	Optional
Configure devices' parallel poll responses	PARALLEL POLL CONFIGURE	Optional
Disable devices' parallel poll capability	PARALLEL POLL UNCONFIGURE	Optional

Table 1. IEEE 488.2 Required and Optional Control Sequences

Keyword	**Name**	**Compliance**
RESET	Reset System	Mandatory
FINDRQS	Find Device Requesting Service	Optional
ALLSPOLL	Serial Poll All Devices	Mandatory
PASSCTL	Pass Control	Optional
REQUESTCTL	Request Control	Optional
FINDLSTN	Find Listeners	Optional
SETADD	Set Address	Optional, but requires FINDLSTN
TESTSYS	Self-Test System	Optional

Table 2. IEEE 488.2 Controller Protocols

IEEE 488.2 Control Sequences – The IEEE 488.2 standard defined control sequences that specify the exact IEEE 488.1 messages that are sent from the Controller as well as the ordering of multiple messages. IEEE 488.2 defined 15 required control sequences and four optional control sequences, as shown in Table 1.

The IEEE 488.2 control sequences describe the exact states of the GPIB and the ordering of command messages for each of the defined operations. IEEE 488.2 control sequences remove the ambiguity of the possible bus conditions, so instruments and Controllers are much more compatible. By exactly defining the state of the bus and how devices should respond to specific messages, IEEE 488.2 removes such system development problems.

IEEE 488.2 Protocols – Protocols are high-level routines that combine a number of control sequences to perform common test system operations. IEEE 488.2 defines two required protocols and six optional protocols, as shown in Table 2. These protocols reduce development time because they combine several commands to execute the most common operations required by any test system. The RESET protocol ensures that the GPIB has been initialized and all devices have been cleared and set to a known state. The ALLSPOLL protocol serial polls each device and returns the status byte of each device.

The PASSCTL and REQUESTCTL protocols pass control of the bus between a number of different devices. The TESTSYS protocol instructs each device to run its own self-tests and report back to the Controller whether it has a problem or is ready for operation.

Mnemonic	Group	Description
*IDN?	System Data	Identification query
*RST	Internal Operations	Reset
*TST?	Internal Operations	Self-test query
*OPC	Synchronization	Operation complete
*OPC?	Synchronization	Operation complete query
*WAI	Synchronization	Wait to complete
*CLS	Status and Event	Clear status
*ESE	Status and Event	Event status enable
*ESE?	Status and Event	Event status enable query
*ESR?	Status and Event	Event status register query
*SRE	Status and Event	Service request enable
*SRE?	Status and Event	Service request enable query
*STB?	Status and Event	Read status byte query

Table 3. IEEE 488.2 Mandatory Common Commands

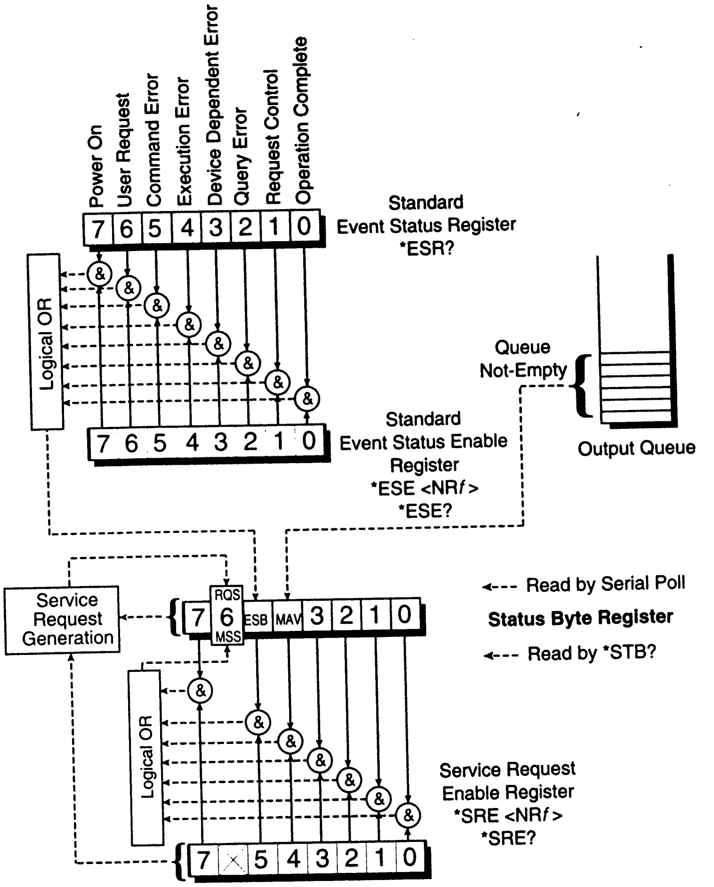


Figure 9. IEEE 488.2 Status Report Model

Perhaps the two most important protocols are FINDLSTN and FINDRQS. The FINDLSTN protocol takes advantage of the IEEE 488.2 Controller capability of monitoring bus lines to locate listening devices on the bus. The Controller implements the FINDLSTN protocol by issuing a particular listen address and then monitoring the NDAC handshake line to determine if a device exists at that address. The result of the FINDLSTN protocol is a list of addresses for all the located devices. FINDLSTN is used at the start of an application program to ensure proper system configuration and to provide a valid list of GPIB devices that can be used as the input parameter to all other IEEE 488.2 protocols. The bus line monitoring capability of an IEEE 488.2 Controller is also useful to detect and diagnose problems within a test system.

The FINDRQS protocol is an efficient mechanism for locating and polling devices that are requesting service. It uses the IEEE 488.2 Controller capability of sensing the FALSE to TRUE transition of the SRQ line. You prioritize the input list of devices so that the more critical devices receive service first. If the application program can jump to this protocol immediately upon the assertion of the SRQ line, you increase program efficiency and throughput.

IEEE 488.2 Instruments

The IEEE 488.2 specifications for instruments can require major changes in the firmware and possibly the hardware. However, IEEE 488.2 instruments are easier to program because they respond to common commands and queries in a well defined manner using standard message exchange protocols and data formats. The IEEE 488.2 message exchange protocol is the foundation for the SCPI standard that makes test system programming even easier.

IEEE 488.2 defines a minimum set of IEEE 488.1 interface capabilities that an instrument must have. All devices must be able to send and receive data, request service, and respond to a device clear message.

IEEE 488.2 defines precisely the format of commands sent to instruments and the format and coding of responses sent by instruments.

All instruments must perform certain operations to communicate on the bus and report status. Because these operations are common to all instruments, IEEE 488.2 defined the programming commands used to execute these operations and the queries used to receive common status information. These common commands and queries are shown in Table 3. Because IEEE 488.2 standardizes status reporting, the Controller knows exactly how to obtain status information from every instrument in the system. This status reporting model builds upon the IEEE 488.1 status byte to provide more detailed status information. The status reporting model is shown in Figure 9.

SCPI

In April 1990, a group of instrument manufacturers announced the SCPI specification, which defines a common command set for programming instruments. Before SCPI, each instrument manufacturer developed its own command sets for its programmable instruments. This lack of standardization forced test system developers to learn a number of different command sets and instrument-specific parameters for the various instruments used in an application, leading to programming complexities and resulting in unpredictable schedule delays and development costs. By defining a standard programming command set, SCPI decreases development time and increases the readability of test programs and the ability to interchange instruments.

SCPI is a complete, yet extendable, standard that unifies the software programming commands for instruments. The first version of the standard was released in mid-1990. Today, the SCPI Consortium continues to add commands and functionality to the SCPI standard. SCPI has its own set of required common commands in addition to the mandatory IEEE 488.2 common commands and queries. Although IEEE 488.2 is used as its basis, SCPI defines programming commands that you can use with any type of hardware or communication link.

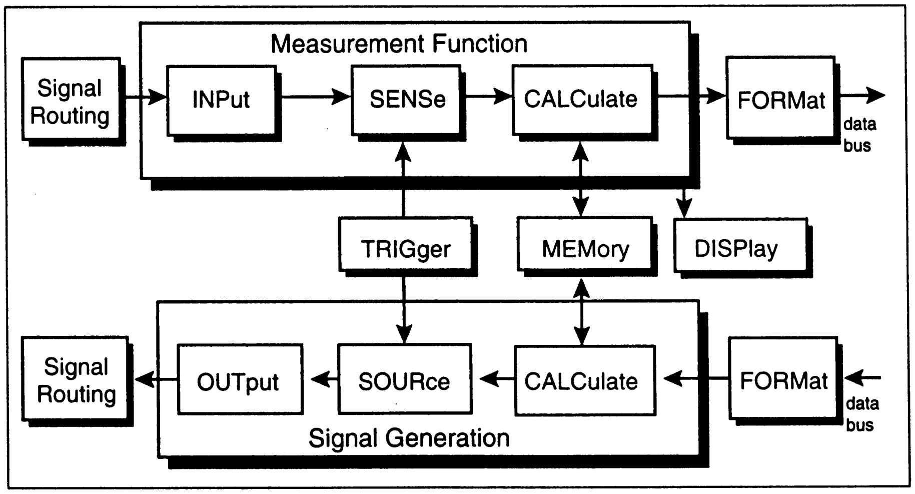


Figure 10. The SCPI Instrument Model

SCPI specifies standard rules for abbreviating command keywords and uses the IEEE 488.2 message exchange protocol rules to format commands and parameters. You may use command keywords in their long form (MEASure) or their short form shown in capital letters (MEAS).

SCPI offers numerous advantages to the test engineer. One of these is that SCPI provides a comprehensive set of programming functions covering all the major functions of an instrument. This standard command set ensures a higher degree of instrument interchangeability and minimizes the effort involved in designing new test systems. The SCPI command set is hierarchical, so adding commands for more specific or newer functionality is easily accommodated.

The SCPI Instrument Model – As a means of achieving compatibility and categorizing command groups, SCPI defined a model of a programmable instrument. This model, shown in Figure 10, applies to all the different types of instrumentation.

All of the functional components of the instrument model may not apply to every instrument. For example, an oscilloscope does not have the functionality defined by the signal generation block in the SCPI model. SCPI defines hierarchical command sets to control specific functionality within each of these functional components.

The signal routing component controls the connection of a signal to the instrument's internal functions; the measurement component converts the signal into a preprocessed form; and the signal generation component converts internal data to real-world signals. The memory component stores data inside the instrument. The format component converts the instrument data to a form that you can transmit across a standard bus. The trigger component synchronizes instrument actions with internal functions, external events, or other instruments.

The measurement function gives the highest level of compatibility between instruments because a measurement is specified by signal parameters, not instrument functionality. In most cases, you can exchange an instrument that makes a particular measurement with another instrument capable of making the same measurement without changing the SCPI command.

The MEASurement component is subdivided into three distinct parts – INPut, SENSe, and CALCulate. The INPut component conditions the incoming signal before it is converted into data by the SENSe block. INPut functions include filtering, biasing, and attenuation. The SENSe component converts signals into internal data that you can manipulate. SENSe functions control such parameters as range, resolution, gate time, and normal mode rejection. The CALCulate component converts the acquired data into a more useful format for a particular application. CALCulation functions include converting units, rise time, fall time, and frequency parameters.

The signal generation component converts data into output as physical signals. SCPI subdivides the signal generation block into three function blocks – OUTPut, SOURce, and CALCulate. The OUTPut block conditions the outgoing signal after it is generated. OUTPut block functions include filtering, biasing, and attenuation. The SOURce block generates a signal based on specified characteristics and internal data. SOURce block functions specify such signal parameters as amplitude modulation, power, current, voltage, and frequency. The CALCulate block converts application data to account for signal generation anomalies such as correcting for external effects, converting units, and changing domains.

Example SCPI Command – The following command programs a digital multimeter (DMM) to configure itself to make an AC voltage measurement on a signal of 20 V with a 0.001 V resolution.

```
: MEASure:VOLTage:AC? 20, 0.001
```

- The leading colon indicates a new command is coming
- The keywords MEASure:VOLTage:AC instruct the DMM to take an AC voltage measurement
- The ? instructs the DMM to return its measurement to the computer/controller
- The 20, 0.001 specifies the range (20 V) and resolution (.001 V) of the measurement

Reference Documents – For more information on the GPIB standards, refer to the following documents – ANSI/IEEE 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation, ANSI/IEEE 488.2-1992, IEEE Codes, Formats, Protocols, and Common Commands, and Standard Commands for Programmable Instruments. You can obtain the IEEE 488 specifications by writing to The Institute of Electrical and Electronic Engineers (IEEE), 345 East 47th St., New York, NY 10017, USA. The latest SCPI Standards are published by the SCPI Consortium.

The High-Speed GPIB Handshake Protocol

National Instruments has developed the patented high-speed GPIB handshake protocol (called HS488) to increase the data transfer rate of a GPIB system.

All devices involved in a data transfer must be HS488 compliant to use the HS488 protocol, but when non-HS488 devices are involved, the HS488 devices automatically use the standard IEEE 488.1 handshake to ensure compatibility.

HS488 is a superset of the IEEE 488 standards. HS488 has been proposed as an addition to the IEEE 488.1 standard and is currently being considered by the Working Group for Higher Performance IEEE 488.1.

IEEE 488 Handshake

The standard IEEE 488.1 3-wire handshake (shown in Figure 7) requires the Listener to unassert Not Ready for Data (NRFD), the Talker to assert the Data Valid (DAV) signal to indicate to the Listener that a data byte is available, and for the Listener to unassert the Not Data Accepted (NDAC) signal when it has accepted that byte. A byte cannot transfer in less than the time it takes for the following events to occur – NRFD to propagate to the Talker, DAV signal to propagate to all Listeners, the Listeners to accept the byte and assert NDAC, the NDAC signal to propagate back to the Talker, and the Talker to allow a settling time (T1) before asserting DAV again.

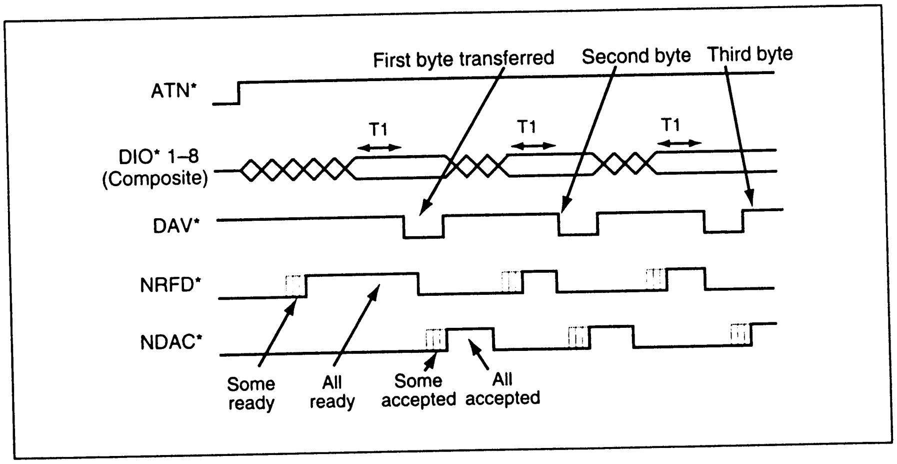


Figure 7. Normal IEEE 488.1 Handshake

HS488 Handshake

HS488 increases system throughput by removing propagation delays associated with the 3-wire handshake. To enable the HS488 handshake, the Talker pulses the NRFD signal line after the Controller addresses all Listeners. If the Listener is HS488 capable, then the transfer occurs using the HS488 handshake (shown in Figure 8). Once HS488 is enabled, the Talker places a byte on the GPIB DIO lines, waits for a preprogrammed settling time, asserts DAV, waits for a preprogrammed hold time, unasserts DAV, and drives the next data byte on the DIO lines. The Listener keeps NDAC unasserted and must accept the byte within the specified hold time. A byte must transfer in the time set by the settling time and hold time, without waiting for any signals to propagate along the GPIB cable.

HS488 Data Transfer Flow Control – The Listener may assert NDAC to temporarily prevent more bytes from being transmitted, or assert NRFD to force the Talker to use the 3-wire handshake. Through these methods, the Listener can limit the average transfer rate. However, the Listener must have an input buffer that can accept short bursts of data at the maximum rate, because by the time NDAC or NRFD propagates back to the Talker, the Talker may have already sent another byte.

The required settling and hold times are user configurable, depending on the total length of cable and number of devices in the system. Between two devices and 2 m of cable, HS488 can transfer data at up to 8 Mbytes/s. For a fully loaded system with 15 devices and 15 m of cable, HS488 transfer rates can reach 1.5 Mbytes/s.

HS488 Controllers always use the standard IEEE 488.1 3-wire handshake to transfer GPIB commands (bytes with Attention (ATN) asserted).

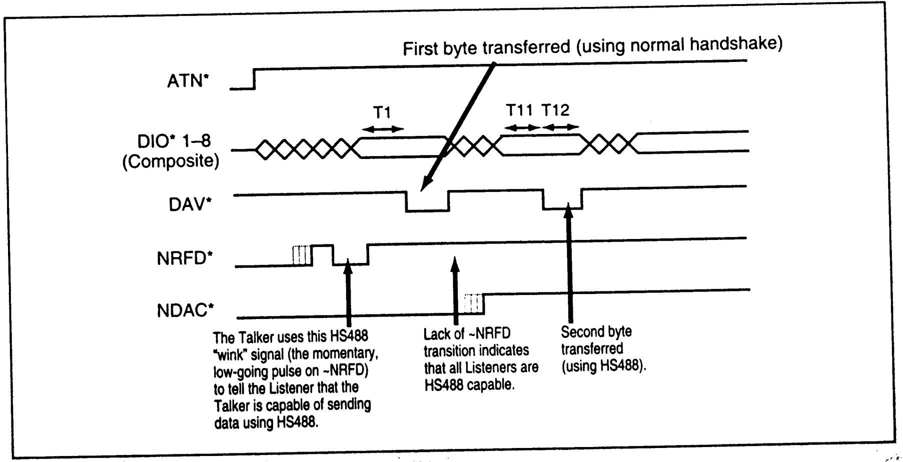

First byte transferred (using normal handshake)
ATN*
T1
T11 T12
DIO* 1–8
(Composite)
DAV*
NRFD*
NDAC*
The Talker uses this HS488 "wink" signal (the momentary, low-going pulse on ~NRFD) to tell the Listener that the Talker is capable of sending data using HS488.
Lack of ~NRFD transition indicates that all Listeners are HS488 capable.
Second byte transferred (using HS488).

Figure 8. HS488 Handshake

Ur: Manual till HP34401A multimeter

An Introduction to the SCPI Language

SCPI (*Standard Commands for Programmable Instruments*) is an ASCII-based instrument command language designed for test and measurement instruments. *Refer to "Simplified Programming Overview," starting on page 112, for an introduction to the basic techniques used to program the multimeter over the remote interface.*

SCPI commands are based on a hierarchical structure, also known as a *tree system*. In this system, associated commands are grouped together under a common node or root, thus forming *subsystems*. A portion of the SENSE subsystem is shown below to illustrate the tree system.

```
SENSe:
   VOLTage:
      DC:RANGe {<range>|MINimum|MAXimum}
   VOLTage:
      DC:RANGe? [MINimum|MAXimum]

   FREQuency:
      VOLTage:RANGe {<range>|MINimum|MAXimum}
   FREQuency:
      VOLTage:RANGe? [MINimum|MAXimum]

   DETector:
      BANDwidth {3|20|200|MINimum|MAXimum}
   DETector:
      BANDwidth? [MINimum|MAXimum]

   ZERO:
      AUTO {OFF|ONCE|ON}
   ZERO:
      AUTO?
```

SENSe is the root keyword of the command, VOLTage and FREQuency are second-level keywords, and DC and VOLTage are third-level keywords. A *colon* (:) separates a command keyword from a lower-level keyword.

Command Format Used in This Manual

The format used to show commands in this manual is shown below:

```
VOLTage:DC:RANGe {<range>|MINimum|MAXimum}
```

The command syntax shows most commands (and some parameters) as a mixture of upper- and lower-case letters. The upper-case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, send the long form.

For example, in the above syntax statement, `VOLT` and `VOLTAGE` are both acceptable forms. You can use upper- or lower-case letters. Therefore, `VOLTAGE`, `volt`, and `Volt` are all acceptable. Other forms, such as `VOL` and `VOLTAG`, will generate an error.

Braces ({ }) enclose the parameter choices for a given command string. The braces are not sent with the command string.

A *vertical bar* (|) separates multiple parameter choices for a given command string.

Triangle brackets (< >) indicate that you must specify a value for the enclosed parameter. For example, the above syntax statement shows the *range* parameter enclosed in triangle brackets. The brackets are not sent with the command string. You must specify a value for the parameter (such as `"VOLT:DC:RANG 10"`).

Some parameters are enclosed in *square brackets* ([ ]). The brackets indicate that the parameter is optional and can be omitted. The brackets are not sent with the command string. If you do not specify a value for an optional parameter, the multimeter chooses a default value.

Command Separators

A *colon* (:) is used to separate a command keyword from a lower-level keyword. You must insert a *blank space* to separate a parameter from a command keyword. If a command requires more than one parameter, you must separate adjacent parameters using a *comma* as shown below:

```
"CONF:VOLT:DC 10, 0.003"
```

A *semicolon* (;) is used to separate commands within the *same* subsystem, and can also minimize typing. For example, sending the following command string:

```
"TRIG:DELAY 1; COUNT 10"
```

... is the same as sending the following two commands:

```
"TRIG:DELAY 1"
"TRIG:COUNT 10"
```

Use a colon *and* a semicolon to link commands from *different* subsystems. For example, in the following command string, an error is generated if you do not use both the colon *and* semicolon:

```
"SAMP:COUN 10;:TRIG:SOUR EXT"
```

Using the *MIN* and *MAX* Parameters

You can substitute MINimum or MAXimum in place of a parameter for many commands. For example, consider the following command:

```
VOLTage:DC:RANGe {<range>|MINimum|MAXimum}
```

Instead of selecting a specific voltage range, you can substitute MIN to set the range to its minimum value or MAX to set the range to its maximum value.

Querying Parameter Settings

You can query the current value of most parameters by adding a *question mark* (**?**) to the command. For example, the following command sets the sample count to 10 readings:

```
"SAMP:COUN 10"
```

You can query the sample count by executing:

```
"SAMP:COUN?"
```

You can also query the minimum or maximum count allowed as follows:

```
"SAMP:COUN? MIN"
"SAMP:COUN? MAX"
```

aution

If you send two query commands without reading the response from the first, and then attempt to read the second response, you may receive some data from the first response followed by the complete second response. To avoid this, do not send a query command without reading the response. When you cannot avoid this situation, send a device clear before sending the second query command.

SCPI Command Terminators

A command string sent to the multimeter *must* terminate with a *<new line>* character. The IEEE-488 *EOI* (end-or-identify) message is interpreted as a *<new line>* character and can be used to terminate a command string in place of a *<new line>* character. A *<carriage return>* followed by a *<new line>* is also accepted. Command string termination will *always* reset the current SCPI command path to the root level.

IEEE-488.2 Common Commands

The IEEE-488.2 standard defines a set of *common commands* that perform functions like reset, self-test, and status operations. Common commands always begin with an asterisk (*), are four to five characters in length, and may include one or more parameters. The command keyword is separated from the first parameter by a *blank space*. Use a *semicolon* (;) to separate multiple commands as shown below:

```
"*RST; *CLS; *ESE 32; *OPC?"
```

SCPI Parameter Types

The SCPI language defines several different data formats to be used in program messages and response messages.

Numeric Parameters Commands that require numeric parameters will accept all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation. Special values for numeric parameters like MINimum, MAXimum, and DEFault are also accepted. You can also send engineering unit suffixes with numeric parameters (e.g., M, K, or u). If only specific numeric values are accepted, the multimeter will automatically round the input numeric parameters. The following command uses a numeric parameter:

```
VOLTage:DC:RANGe {<range>|MINimum|MAXimum}
```

Discrete Parameters Discrete parameters are used to program settings that have a limited number of values (like BUS, IMMediate, EXTernal). They have a short form and a long form just like command keywords. You can mix upper- and lower-case letters. Query responses will *always* return the short form in all upper-case letters. The following command uses discrete parameters:

```
TRIGger:SOURce {BUS|IMMediate|EXTernal}
```

Boolean Parameters Boolean parameters represent a single binary condition that is either true or false. For a false condition, the multimeter will accept "OFF" or "0". For a true condition, the multimeter will accept "ON" or "1". When you query a boolean setting, the instrument will *always* return "0" or "1". The following command uses a boolean parameter:

```
INPut:IMPedance:AUTO {OFF|ON}
```

String Parameters String parameters can contain virtually any set of ASCII characters. A string *must* begin and end with matching quotes; either with a single quote or with a double quote. You can include the quote delimiter as part of the string by typing it twice without any characters in between. The following command uses a string parameter:

```
DISPlay:TEXT <quoted string>
```

Output Data Formats

Output data will be in one of formats shown in the table below.

Type of Output Data	Output Data Format
Non-reading queries	< 80 ASCII character string
Single reading (IEEE-488)	SD.DDDDDDDDESDD<nl>
Multiple readings (IEEE-488)	SD.DDDDDDDDESDD,...,...,<nl>
Single reading (RS-232)	SD.DDDDDDDDESDD<cr><nl>
Multiple readings (RS-232)	SD.DDDDDDDDESDD,...,...,<cr><nl>

S	Negative sign or positive sign
D	Numeric digits
E	Exponent
<nl>	newline character
<cr>	carriage return character

2. VXI

Application Note 030

Short Tutorial on VXI/MXI

Ron Wolfe

The purpose of this application note is to help you gain an understanding of VXI and MXI concepts. This application note is divided into two tutorial sections. The first section discusses VXI and the second section discusses MXI.

VXI Tutorial

This section contains an overall introduction to VXI (VMEbus eXtensions for Instrumentation).

What is VXI?

VXIbus is an exciting and fast-growing platform for instrumentation systems. The VXI Consortium was formed in 1987 with a charter of defining a multivendor instrument-on-a-card standard. Since that time, the Consortium has defined system-level components required for hardware interoperatibility. The IEEE officially adopted the VXI specification, IEEE 1155, in March 1993. The VXI*plug&play* Systems Alliance, founded in September 1993, sought a higher level of system standardization to cover all VXI system components. By focusing on software standardization, the alliance defined standards to make VXI systems easy to integrate and use while maintaining multivendor software interoperatibility. With the success of multivendor standards and solid technical specifications, VXI is backed by more than 250 vendors, with more than 1000 products available. The success of VXI as an open, multivendor platform is a testament to the value of multivendor standards, and has made VXI the platform of choice for open instrumentation systems.

VXI is used in many different applications ranging from test and measurement and ATE, to data acquisition and analysis in both research and industrial automation. Although some VXI systems today are purely VXI, many users are migrating to VXI by integrating it into existing systems consisting of GPIB instruments, VME cards, or plug-in data acquisition (DAQ) boards. You can control a VXI system with a remote general-purpose computer using the high-speed Multisystem eXtension Interface (MXI) bus interface or GPIB. You can also embed a computer into a VXI chassis and control the system directly. Whatever your system configuration needs may be, VXI offers the flexibility and performance to take on today's most challenging applications.

The Need for VXIbus

The demand for an industry-standard instrument-on-a-card architecture has been driven by the need for physical size reduction of rack-and-stack instrumentation systems, tighter timing and synchronization between multiple instruments, and faster transfer rates than the 1 Mbytes/s rate of the 8-bit GPIB. The modular form factor, high bandwidth, and commercial success of the VMEbus made it particularly attractive as an instrumentation platform. The tremendous popularity of GPIB also made it attractive as a model for device communication and instrument control protocols. The VXIbus specification adds the standards necessary to combine the VMEbus with GPIB to create a new, modular instrumentation platform that can meet the needs of future instrumentation applications.

Product and company names are trademarks or trade names of their respective companies.

VXI brings the following benefits to instrumentation users:

- Open, multivendor standards maximize flexibility and minimize obsolescence
- Increased system throughput reduces test time and/or increases capabilities
- Smaller size and higher density reduce floor space, enhance mobility or portability, and give close proximity to devices(s) being tested or controlled
- More precise timing and synchronization improve measurement capability
- Standardized VXI*plug&play* software eases system configuration, programming, and integration
- Modular, rugged design improves reliability, increases mean-time between failure (MTBF), and decreases mean-time to repair (MTTR)

Members of the VXIbus Consortium and the VXI*plug&play* Systems Alliance have combined their expertise to develop technically sound standards for both hardware and software, bringing the entire industry into a new generation of easy-to-use instrumentation.

The Value of Open Industry Standards

The baseline VXI hardware specifications are a mandate for interoperatibility between hardware products from different vendors. These specifications cover mechanical and environmental requirements such as module sizes, mainframe and module cooling, and EMC compatibility between modules, as well as automated system initialization and backplane communication protocols. The VXI*plug&play* Systems Alliance builds on these baseline specifications to address the system as a whole with the goal of having the end-user up and running in “Five minutes or less.” Building a system based on open industry standards means that you choose components for your system based on your requirements, regardless of vendor. Open standards also ensure that once your system is built, your investment will continue to pay dividends well into the future.

Both the VXI Consortium and the VXI*plug&play* Systems Alliance remain strong, active organizations committed to maintaining VXI as an open, multivendor technology and increasing its ease of use and end-user success. In fact, many of the largest instrument suppliers in the world are members of both organizations, including National Instruments, GenRad, Hewlett-Packard, Racal Instruments, and Tektronix. With VXI*plug&play*, you are assured that components from different vendors will work reliably in the same system. Members of the VXI Consortium and the VXI*plug&play* Systems Alliance have combined their expertise to develop technically sound standards for both hardware and software, bringing the entire industry into a new generation of instrumentation – a generation that stresses ease of use and open systems without sacrificing flexibility or performance.

VXIbus Mechanical Configuration

Physically, a VXIbus system consists of a mainframe chassis that has the physical mounting and backplane connections for plug-in modules, as shown in Figure 1. The VXIbus uses the industry-standard IEEE-1014 VMEbus as a base architecture to build upon. As shown in Figure 2, VXI uses the full 32-bit VME architecture, but adds two board sizes and one connector. The P1 connector and the center row of the P2 connector are retained exactly as defined by the VME specification. The VME user-definable pins on the P2 connector and the additional pins on P3, the third VXI connector, implement instrumentation signals between plug-in modules directly on the backplane.

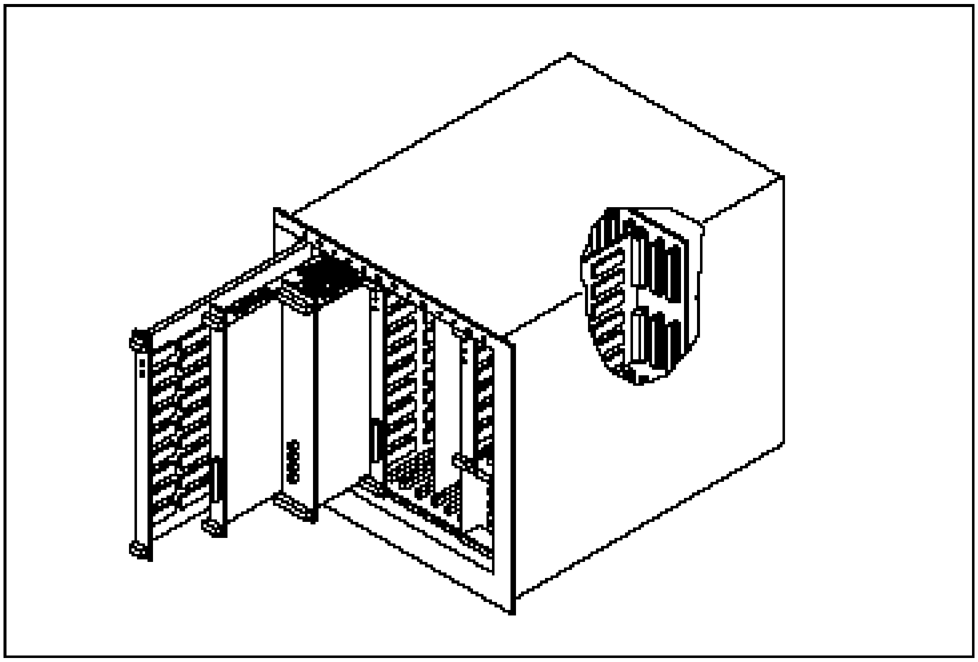


Figure 1. A VXIbus System

The VXIbus specification includes packaging requirements, electromagnetic compatibility, power distribution, cooling and air flow for VXIbus mainframes and plug-in modules. The modules are installed in the mainframe slots. LEDs, switches, test points, and I/O connections are accessible from the module front panel.

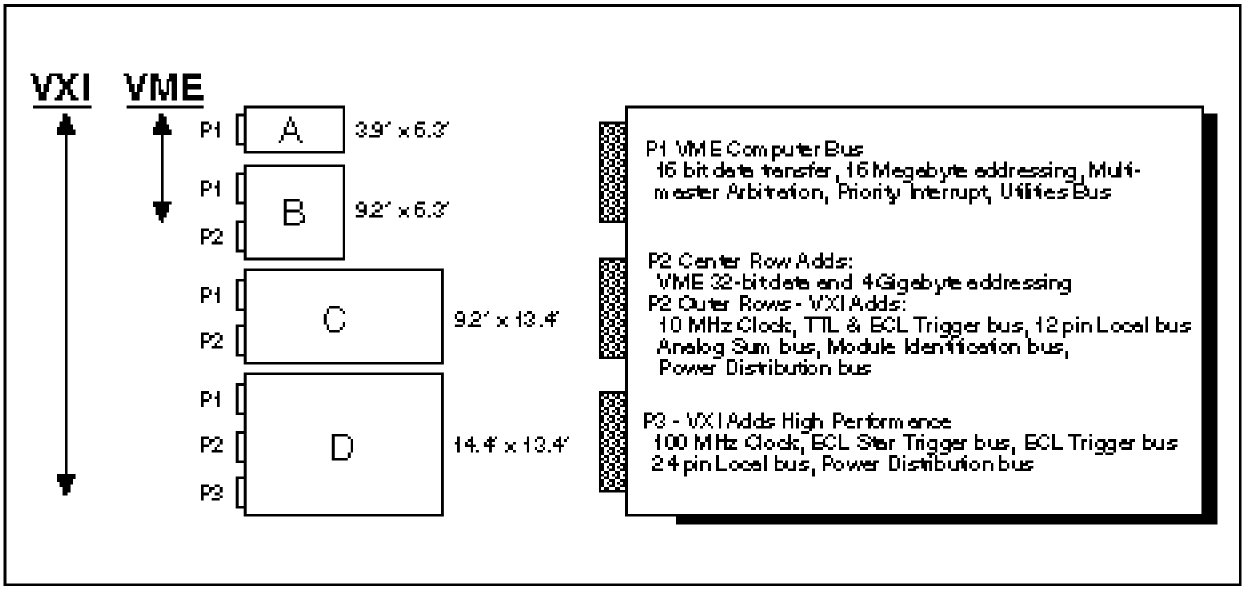

VXI
VME
P1
A
3.9" x 6.3"
P1
B
P2
9.2" x 6.3"
P1
C
P2
9.2" x 13.4"
P1
P2
D
P3
14.4" x 13.4"
P1 VME Computer Bus
16 bit data transfer, 16 Megabyte addressing, Multi-master Arbitration, Priority Interrupt, Utilities Bus
P2 Center Row Adds:
VME 32-bit data and 4Gigabyte addressing
P2 Outer Rows - VXI Adds:
10 MHz Clock, TTL & ECL Trigger bus, 12 pin Local bus
Analog Sum bus, Module Identification bus,
Power Distribution bus
P3 - VXI Adds High Performance
100 MHz Clock, ECL Star Trigger bus, ECL Trigger bus
24 pin Local bus, Power Distribution bus

Figure 2. VXI Module Sizes

Module and Mainframe Cooling

Airflow direction is from bottom (P3) to top (P1). Cooling requirements must be established for all modules and included in product specifications. These requirements must include an operating point of minimum airflow requirement. Mainframe suppliers must also provide similar information for their mainframes.

EMC and Noise

The addition of a new module to a VXIbus system must not degrade the performance of any other module. The VXIbus specification includes near-field radiation and susceptibility requirements, which prevent one module from interfering with the operation of other modules. To help meet these requirements, the VXIbus module width was increased from the 0.8 in. VME requirement to 1.2 in., so that there is enough room for the module to be completely enclosed in a metal case for shielding. The metal cases connect to backplane grounds. Thus, you can use existing VME boards in a VXIbus chassis, but not vice versa.

The VXIbus specification also has conducted-emissions and susceptibility requirements, which prevent any power supply noise from affecting the performance of a module. For far-field radiated emissions such as FCC and VDE, each module must not contribute more than its share of the total. For example, in a mainframe that holds 13 modules, each module must not contribute more than 1/13 of the allowed total. Because of the desire for extremely precise time coupling between modules using the backplane, it is necessary to minimize the noise and crosstalk on the backplane clock and trigger signal lines. The backplane is required to be a single, monolithic board across any one slot. The VXIbus specification has a tutorial section on how to design a backplane for low noise and high signal integrity.

Hardware Registers

VXI modules must have a specific set of registers located at specific addresses, as shown in Figure 3. The upper 16 KB of the 64 KB A16 address space are reserved for VXIbus devices. Each VXI device has an 8-bit logical address that specifies where its registers are located in this address space. A single VXI system can have up to 256 VXI devices. The logical address of a VXI device, which can be manually set or automatically configured by the system at startup, is analogous to the GPIB address of a GPIB device.

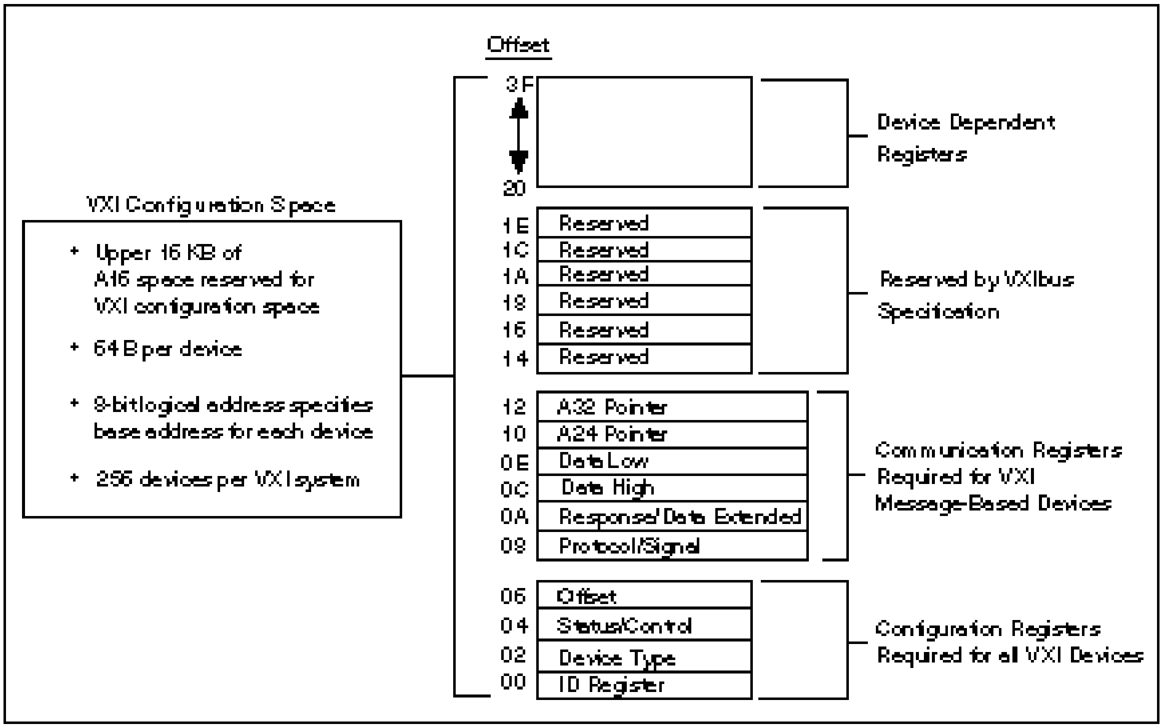

Offset
VXI Configuration Space
• Upper 16 KB of A16 space reserved for VXI configuration space
• 64 B per device
• 8-bit logical address specifies base address for each device
• 256 devices per VXI system
3F
20
Device Dependent Registers
1E Reserved
1C Reserved
1A Reserved
18 Reserved
16 Reserved
14 Reserved
Reserved by VXIbus Specification
12 A32 Pointer
10 A24 Pointer
0E Data Low
0C Data High
0A Response/Data Extended
08 Protocol/Signal
Communication Registers Required for VXI Message-Based Devices
06 Offset
04 Status/Control
02 Device Type
00 ID Register
Configuration Registers Required for all VXI Devices

Figure 3. VXI Configuration Registers

Register-Based Devices

Because of the VXI configuration registers, which are required for all VXI devices, the system can identify each VXI device, its type, model and manufacturer, address space, and memory requirements. VXIbus devices with only this minimum level of capability are called Register-Based devices. With this common set of configuration registers, the centralized Resource Manager (RM), essentially a software module, can perform automatic system and memory configuration when the system is initialized.

Message-Based Communication

In addition to Register-Based devices, the VXIbus specification also defines Message-Based devices, which are required to have communication registers and configuration registers. All Message-Based VXIbus devices, regardless of the manufacturer, can communicate at a minimum level using the VXI-specified Word Serial Protocol. When minimum communication is possible, higher-performance communication channels, such as shared-memory channels, can be established to take advantage of the VXIbus bandwidth capabilities.

Word Serial Protocol

The VXIbus Word Serial Protocol is functionally very similar to the IEEE-488 protocol, which transfers data messages to and from devices one byte (or word) at a time. Thus, VXI Message-Based devices communicate in a fashion very similar to IEEE-488 instruments. In general, Message-Based devices typically contain some level of local intelligence that uses or requires a high level of communication.

All VXI Message-Based devices are required to use the Word Serial Protocol to communicate in a standard way. The protocol is called *word serial*, because if you want to communicate with a Message-Based device, you do so by writing and reading 16-bit words one at a time to and from the Data In (write Data Low) and Data Out (read Data Low) hardware registers located on the device itself. Word Serial communication is paced by the bits in the response register of the device, indicating whether the Data In register is empty and whether the Data Out register is full. This operation is very similar to Universal Asynchronous Receiver Transmitter (UART) on a serial port.

Commander/Servant Hierarchies

The VXIbus defines a Commander/Servant communication protocol so you can construct hierarchical systems using conceptual layers of VXI devices. This structure is like an inverted tree. A Commander is any device in the hierarchy with one or more associated lower-level devices, or Servants. A Servant is any device in the subtree of a Commander. A device can be both a Commander and a Servant in a multiple-level hierarchy.

A Commander has exclusive control of the communication and configuration registers of its immediate Servants (one or more). Any VXI module has one and only one Commander. Commanders communicate with Servants through the communication registers of the Servants using the Word Serial Protocol if the Servant is a Message-Based device, or by device-specific register manipulation if the Servant is a Register-Based device. Servants communicate with their Commander by responding to the Word Serial commands and queries from their Commander through the Word Serial protocol if they are Message-Based, or by device-specific register status if they are Register-Based.

Interrupts and Asynchronous Events

Servants can communicate asynchronous status and events to their Commander through hardware interrupts or by writing specific messages (signals) directly to their Commander's hardware Signal Register. Nonbusmaster devices always transmit such information via interrupts, whereas devices that have busmaster capability can either use interrupts or send signals. Some Commanders can receive signals only, whereas others might be only interrupt handlers.

The VXIbus specification contains defined Word Serial commands so that a Commander can understand the capabilities of its Message-Based Servants and configure them to generate interrupts or signals in a particular way. For example, a Commander can instruct its Servants to use a particular interrupt line, to send signals rather than generate interrupts, or configure the reporting of only certain status or error conditions.

Although the Word Serial Protocol is reserved for Commander/Servant communications, peer-to-peer communication between two VXI devices can be established through a specified shared-memory protocol or by simply writing specific messages directly to the signal register of the device.

Slot 0 and the Resource Manager

The leftmost slot of a VXI chassis has special system resources such as backplane clocks, configuration signals, and synchronization (trigger) signals and therefore must be occupied by a device with VXI "Slot 0" capabilities. The VXI Resource Manager (RM) function, essentially a software module, can reside on any VXI module or even on an external computer. The RM, in combination with the Slot 0 device, identifies each device in the system, assigns logical addresses, memory configurations, and establishes Commander/Servant hierarchies using the Word Serial Protocol to grant Servants to the Commanders in the system. After establishing the Commander/Servant hierarchy, the RM issues the Begin Normal Operation Word Serial command to all top-level Commanders. During normal system operation, the RM may also halt the system and/or remap the hierarchy if necessary.

Three Ways to Control a VXI System

System configuration is divided into three categories. The first type of VXI system consists of a VXI mainframe linked to an external controller via the GPIB. The controller talks across the GPIB to a GPIB-VXI interface module installed inside the VXI mainframe. The GPIB-VXI interface transparently translates the GPIB protocol to and from the VXI Word Serial protocol.

The second configuration involves a VXI-based embedded computer. The embedded computer is a VXI module that resides inside the VXI mainframe and connects directly to the VXI backplane. This configuration offers the smallest physical size for a VXI system as well as performance benefits due to direct connection to the VXI backplane.

The third configuration uses a high-speed MXIbus link from an external computer to control the VXI backplane. The external computer operates as though it is embedded directly inside the VXI mainframe. This configuration is functionally equivalent to the embedded method, except that it has the flexibility for use with a wide variety of computers and workstations.

VXI Bus Interface Software

One of the most important considerations when selecting a VXI system is software. Software is the key to developing successful systems based on the VXIbus. There are many programming languages, operating systems, and application development environments (ADE) to choose from when building a VXI system. It is important to make the right decisions to realize all of the advantages that VXI has to offer, while minimizing your development costs now and in the future.

Your software decisions not only affect overall system performance and system capability, but also development time and productivity. You should choose tools that have complete debugging capability and that work with the most popular operating systems and programming languages. If you choose to program your VXI system using a standard language such as C, C++, Basic, ADA, or ATLAS, you should realize that standard programming languages do not come with built-in VXI capability. Rather, VXI capability is added through a VXI bus interface software library. This software component is very important, because it affects the choice of VXI computer hardware, operating system, programming language, and ADE.

Industry-Wide Software Standards

As a step toward industry-wide software compatibility, the VXI*plug&play* alliance developed one specification for I/O software – the Virtual Instrument System Architecture, or VISA. The VISA specification, VPP-4.1, defines a next-generation I/O software standard not only for VXI, but also for GPIB and serial interfaces. With the VISA standard endorsed by over 50 of the largest instrumentation companies in the industry including Tektronix, Hewlett-Packard, and National Instruments, VISA unifies the industry by facilitating the development of interoperable and reusable software components able to stand the test of time. Before VISA, there were many different commercial implementations of I/O software for VXI, GPIB, and serial interfaces; however, none of these I/O software products were standardized or interoperable.

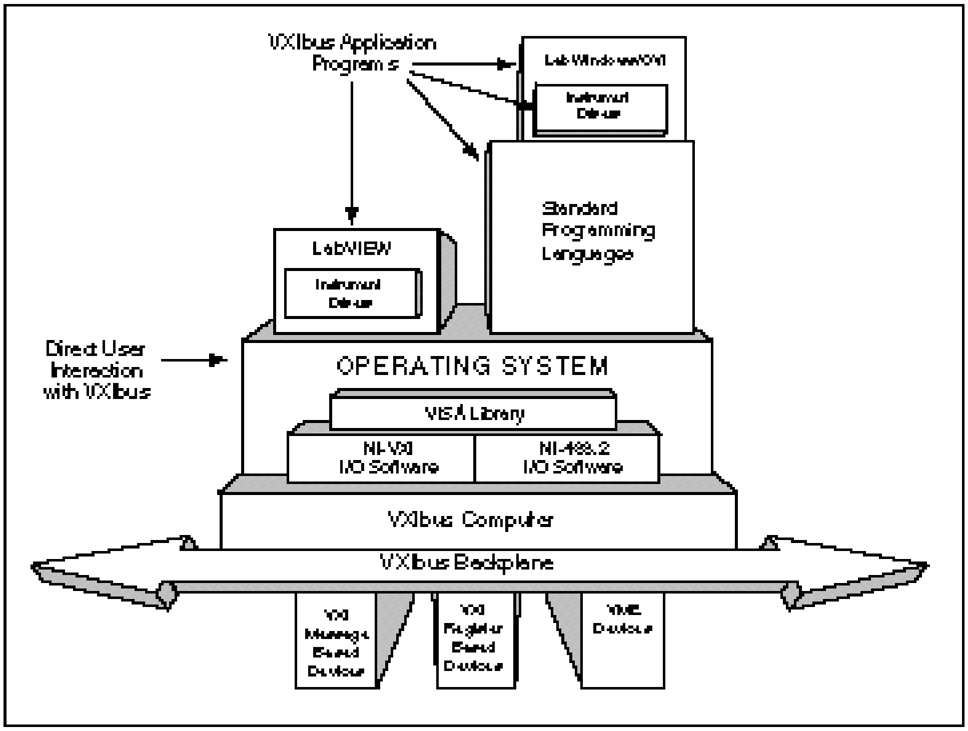


Figure 4. VXIbus System Software Components

The VISA standard lays the foundation and provides a unified migration path for industry-wide software compatibility. One of the most notable benefits of VISA is its ability to significantly reduce the time and effort involved in programming different I/O interfaces. Instead of using a different Application Programmers Interface (API) devoted to each interface bus, you can use the VISA API regardless of whether your system is controlled by GPIB, VXI, or a GPIB-VXI.

With the vast number of choices in instrumentation and software now available, most users do not want to be locked into a specific vendor for their systems. Instead, they would prefer the freedom to select the best instruments and software available from multiple vendors and have it all work together with minimal effort. The IEEE 488.1 and IEEE 488.2 standards (for GPIB) and the IEEE 1155 standard (for VXI) ensured that the hardware would be interoperable, but this approach was not taken for the software. Therefore, the ideal new driver architecture should be a standard adopted by as many of the major vendors as possible. Then you could be assured that any code written for your instrument is portable across controller vendors as well as operating systems. This is exactly what the VXI*plug&play* Systems Alliance has done with VISA.

MXI Tutorial

This section contains an overall introduction to MXI.

MXIbus Overview

The MXI bus is a powerful, high-speed communication link that interconnects devices using a flexible cabling scheme. Derived from the VMEbus, MXI provides a high-performance way of controlling VXI systems using commercially available desktop computers and workstations. National Instruments developed and published the MXI specification and released it as an open industry standard in 1989. In 1995, National Instruments introduced MXI-2, which offers even higher performance.

A MXIbus system configuration combines the performance benefits of a custom embedded VXI computer with the flexibility and availability of general-purpose computers. The MXIbus system configuration uses the high-speed MXIbus cable to connect an external computer directly to the VXI backplane. With the MXIbus, you can locate the computer directly next to the VXI mainframe, or up to 20 meters away. Using the MXIbus, you can easily add other VXI mainframes, and use the plug-in slots in the external computer for GPIB-control, plug-in DAQ boards, or other peripheral adapter cards.

For instrument control, MXI complements high-speed platforms such as PCI by harnessing their high-throughput potential. PCI-based desktop PCs compete with the most advanced computer workstations to provide a low-cost platform that delivers superior performance. You can use low-cost desktop computers to control sophisticated VXI instrumentation without sacrificing performance or control. More importantly, as new desktop computers incorporate the latest technology including faster, more capable microprocessors and RAM, you can easily upgrade your VXI system as these newer and faster computers emerge to immediately reap increased VXI performance gains. Thus, a PCI-based MXI-2 solution such as our VXI-PCI8000 gives you excellent performance now with headroom for the future.

A New Generation of VXI Connectivity

Many VXI users migrate from GPIB-based systems. As a result, the National Instruments GPIB-VXI is a popular way to control VXI instruments from a GPIB controller. An increasingly popular way to control VXI, however, is to use a custom VXI computer that plugs directly into the VXI mainframe, such as the National Instruments VXIpc™ and VXIcpu™ Series of embedded VXI computers. This *embedded* approach is technically attractive because the computer communicates directly with the VXIbus and is tightly coupled to the instruments.

Although an embedded computer is very powerful, custom VXI computers cannot possibly keep pace with the general-purpose computer market. In the last decade, specialized instrument controllers have rapidly declined. General-purpose PCs and workstations, with their vast array of software and accessories, have revolutionized our industry. By using general-purpose computers, the instrumentation industry directly benefits from the billions of R&D dollars spent each year in the general computer market.

Most VXI users would prefer to use an industry-standard computer provided by a computer vendor rather than a VXI-specific computer provided by an instrument vendor. In fact, for VXI to truly become the platform for the next millennium, it must align itself with the powerful general computer market. Then VXI can take advantage of the billions of dollars being spent and bring this investment to bear on the needs of the instrumentation community. VXI must be able to take full advantage of industry-standard PCs with PCI, EISA, and ISA, as well as workstations from Sun, HP, and others. VXI also must have a transparent mechanism for extending to multiple mainframes, and a way to accommodate instruments that cannot physically fit on a VXI module. MXIbus meets each of these needs.

The Need for MXIbus

Today’s market demands that you add value to our Test & Measurement systems. You need modular testing systems that can evolve with technological innovations in our industry. You want increased data throughput and the utmost in computing power; you want flexible, high-speed connectivity between multiple VXI/VME mainframes; and you want to be able to keep up with innovations in PC and workstation technology. Today, sophisticated I/O architectures such as PCI are accelerating data throughput – who knows what tomorrow may hold. How can you take these benefits both now, and in the future? We believe that the answer is MXI.

MXI provides you with a solution that combines the performance benefits of an embedded VXI computer with the flexibility of a general purpose desktop computer. Our VXI-PCI8000 controller and our next generation MXI-2 provides you with an ultra high-performance VXI connectivity solution that can meet your needs both today and well into the future. Although traditional connectivity solutions have proved to be very effective, they also have proved to be the bottleneck in VXI test systems because the software protocol overhead associated with these methods significantly reduced the achievable throughput on the link. Using MXI, this bottleneck is eliminated altogether because MXI devices are connected at the hardware level by mapping each physically separate system into a shared memory space. Physically separate devices transparently share resources through simple reads and writes to the appropriate address in memory. Our next generation MXI-2 products enhance VXI connectivity by defining a single memory-mapped backplane-on-a-bus that can transparently extend bus-level I/O, VXI triggering, interrupts, and systems clocks between systems. You can now use a single cable to conveniently share trigger and timing information between mainframes in a multiple mainframe configuration. The MXI 2.0 specification also defines a synchronous data transfer method that increases MXIbus throughput for block data transfers. From a system standpoint, this means that MXI throughput rates can easily keep up with the data rates of high-performance computers, peripherals, and instrumentation . From a user standpoint, this translates to increased performance and reduced time to test. By choosing a PC-based MXI approach, you are choosing to add value to your VXI instrumentation systems by leveraging technologies that make sense from both a cost and performance perspective.

MXIbus Applications

You can use MXIbus for a variety for a variety of applications. You can interface industry-standard desktop computers to VXIbus or VMEbus; you can create multiple chassis configurations using our VXI-MXI or VME-MXI extenders; and you can integrate VXI and VME chassis into the same test system.

Figures 5 and 6 show two common configurations with MXIbus.

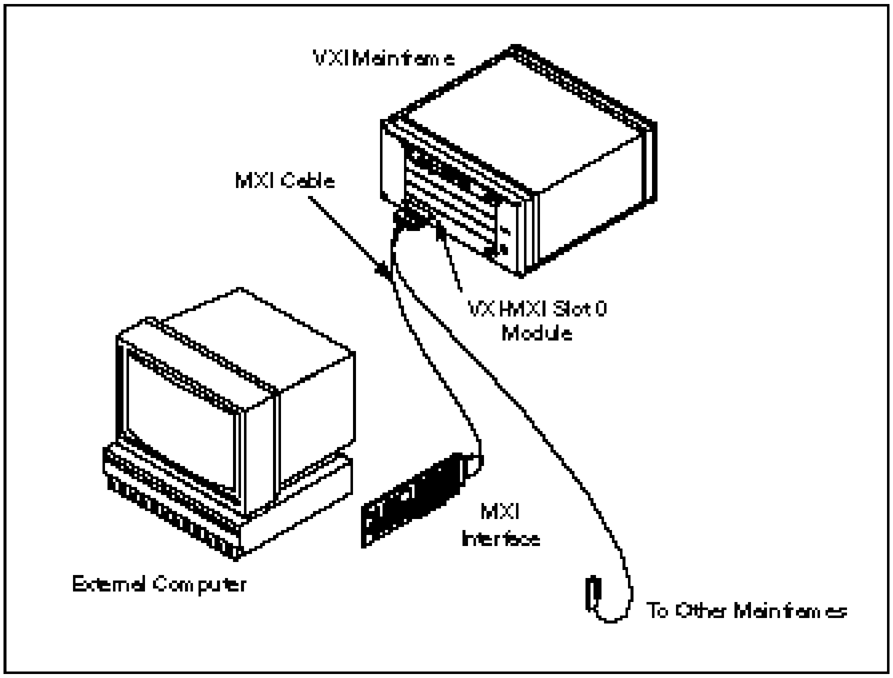


Figure 5. PC Using MXI to Control a VXIbus System

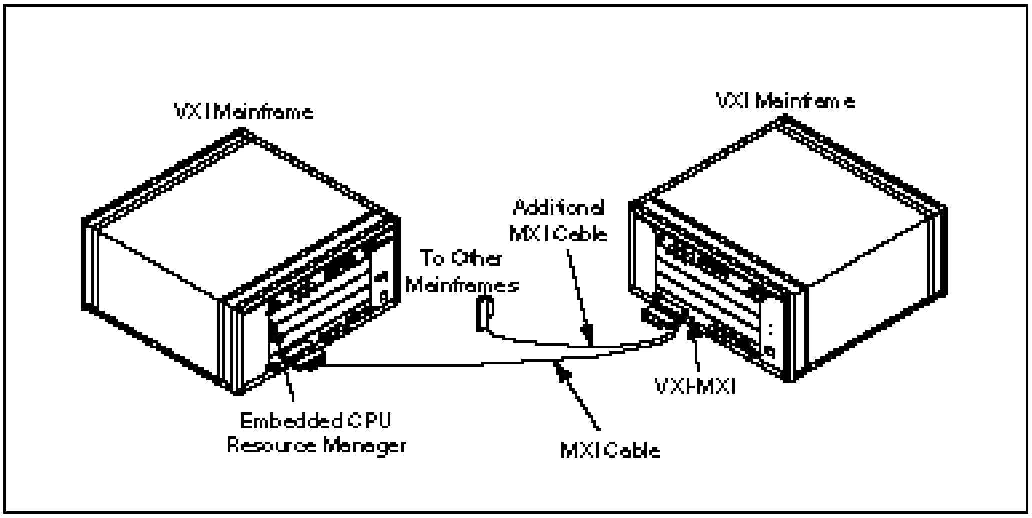


Figure 6. MXI Used for Multiple Mainframe VXI System

How Does MXIbus Work?

MXIbus is a general purpose, 32-bit multimaster system bus on a cable. MXI interconnects multiple devices using a flexible cabling method similar to GPIB, but uses a hardware memory-mapped communication scheme that eliminates the software overhead. MXI devices can directly access each other's resources by performing simple read and writes to appropriate address locations. The new MXI-2 standard expands on the MXI-1 standard by exporting all VXI backplane signals such as VXI-defined trigger lines, interrupt lines and system clocks, in addition to the standard MXIbus signals directly to the cabled bus. MXI-2 users can accomplish critical timing and synchronization tasks between up to eight, daisy-chained MXI devices.

MXI device connectivity is accomplished at the hardware level. The MXI cable serves as a transparent link that interconnects multiple MXI devices. These devices are interlaced by mapping together portions of their individual address spaces so that a system composed of multiple devices behaves as a single system with a shared address space. Figure 7 shows the MXIbus hardware memory-mapped communication. The immediate benefit of this approach is increased data throughput due to the absence of software overhead.

Each MXIbus hardware interface has *address window* circuitry that detects internal (local) bus cycles that map out to the MXIbus. In addition, this circuitry also detects external (remote) MXIbus cycles of connected devices whose addresses map into the shared memory space of the overall system. When a hardware write or read occurs with an address that maps across MXI, the MXI hardware interlocks the bus cycle between the devices via the MXIbus. This hardware scheme is the same as that used by embedded VXI controllers.

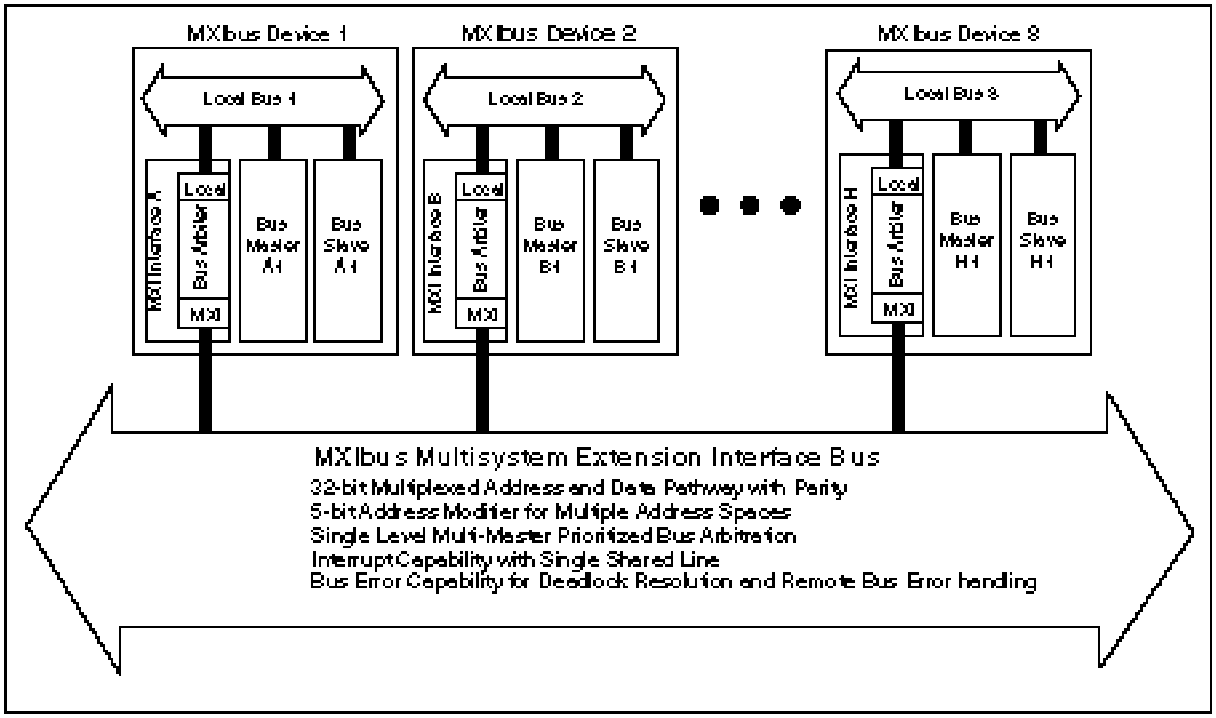

MXIbus Device 1
MXIbus Device 2
MXIbus Device 8
Local Bus 1
Local Bus 2
Local Bus 8
MXI Interface A
MXI Interface B
MXI Interface H
Local
Bus Arbiter
MXI
Bus Master A1
Bus Slave A1
Bus Master B1
Bus Slave B1
Bus Master H1
Bus Slave H1
MXIbus Multisystem Extension Interface Bus
32-bit Multiplexed Address and Data Pathway with Parity
5-bit Address Modifier for Multiple Address Spaces
Single Level Multi-Master Prioritized Bus Arbitration
Interrupt Capability with Single Shared Line
Bus Error Capability for Deadlock Resolution and Remote Bus Error handling

Figure 7. MXIbus Hardware Memory-Mapped Communication

MXIbus Signals

MXIbus signals include 32 multiplexed address and data lines with parity, address modifiers for multiple address spaces, single-level multimaster prioritized bus arbitration, a single interrupt line, a bus error line for handling timeouts and deadlock conditions, and handshake lines for asynchronous operation. Data transfers of 8, 16, and 32 bits are possible, as well as invisible read/write operations and integrated block-mode transfers. With synchronous MXI, the MXI-2 product line can achieve burst data rates as high as 33 Mbytes/s, and sustained throughput rates exceeding 20 Mbytes/s, regardless of the length of the MXI-2 cable

MXIbus Cables

A single MXI cable can be any length up to 20 m. Up to eight MXI devices can be daisy-chained on a single MXI cable length. If multiple MXI devices are daisy-chained together, the total cable distance must be no more than 20 m. The MXI-1 cable is a flexible, round cable similar to a GPIB cable (about 0.6 in. in diameter). Internally there are 48 single-ended, twisted-pair signal lines. MXI-2 features an improved cabling scheme that uses a single double-shielded cable between all devices, and a single high-density, high reliability 144-pin connector per device. In this fashion, all MXI-2 devices share not only the MXIbus itself, but also the VXI-defined trigger lines, interrupt lines, systems clocks, and other signals that were available on MXI-1 products as an optional second connector and cable (INTX). MXI-1 products use an MXI-1 cable between devices, and an optional INTX cable to share trigger/timing information between mainframes in a multiple mainframe configuration. MXI-2 eliminates the need for an additional INTX cable in your system. Because of the cables differences, you cannot mix MXI-1 and MXI-2 products in the same system. Both MXI-1 and MXI-2 use double shielding with an aluminum mylar shield as well as a copper braid shield to eliminate any EMI problems, and both cables meets the National Electric Code (NEC) CL2 fire safety code. The stacking depth of two daisy-chained MXI cables is approximately 3.3 in.

MXI is essentially a backplane bus in a cable. Each MXI signal line is twisted with its own ground line. All MXI signal lines are matched impedance to minimize signal skew and reflections. Stub lengths no more than 4 in. off the mainline interconnection minimize reflections due to impedance discontinuities. Termination networks, configured with onboard jumpers, are located at the first and last MXI devices to minimize reflections at the ends of cables.

MXI uses state-of-the-art, single-ended, trapezoidal bus transceivers to reduce noise crosstalk in the transmission system. Designed specifically for driving backplane bus signals, these transceivers have open-collector drivers that generate precise trapezoidal waveforms with typical rise and fall times of 9 ns. The trapezoidal shape, due to the constant rise and fall times, reduces noise coupling (crosstalk) on adjacent lines. The receiver uses a lowpass filter to remove noise and a high-speed comparator that recognizes the trapezoidal-shaped signal from the noise.

Performance Isssues

MXIbus Performance

It is often difficult to understand how a performance specification for a single component relates to the overall performance of your system. In the case of MXI, it is important to understand not only the performance issues associated with the MXI link, but also the devices that communicate across the link. MXI works like an embedded computer, using a direct hardware memory-map to eliminate software overhead between your computer and the VXIbus or VMEbus. Both MXI and embedded VXI computers can use shared-memory communication protocols and direct register accesses for potentially dramatic performance improvements over GPIB. If your VXI instruments themselves do not use these capabilities, however, your system performance using MXI or an embedded computer may be no higher than a GPIB-controlled VXI system.

There are several factors to consider when comparing a MXI-equipped computer to an embedded computer. A MXI-equipped computer is functionally equivalent to an embedded computer. In fact, application software developed on a MXI computer using NI-VXI/VISA bus interface software can easily run on an embedded computer and vice versa. There are subtle hardware timing differences, but there is no dramatic performance difference due to architecture. MXI, for example, can take roughly 100 ns more to perform a single VXI read or write than an embedded computer, because the MXI signals must propagate down the MXI cable at 10 ns/m. This subtle detail is measured in ns, and is negligible compared to the other factors that affect your system performance, such as the execution speed of your application software or your instruments.

Often, the most important performance issue to consider when evaluating a computer for your system is the performance of the processor itself. Most applications spend much more time computing, displaying, or performing disk I/O than actually performing I/O across the VXIbus or VMEbus. Current external MXI computers are over four times as fast as the fastest embedded VXI computer. In addition, because of the physical space constraints of embedded computers, external computers often have much more sophisticated architectures with faster processors, cache RAM, faster disk drives, and other benefits. Raw computing power can be the single most important consideration for the performance of your system.

Data Transfer Rates

A common benchmark for VXI computers is the Block Data Rate. This benchmark is easy for vendors to isolate and measure under ideal conditions. It is important to understand what Block Data Rate means to your application. Block Data Rate is the rate at which you can move a large block of data to or from memory on an ideal VXI device using back-to-back VXI transfers. It does not measure how fast the computer can process the blocks of data or store them to disk once they are moved, or whether your instruments themselves can actually match that data rate. Most applications are not limited by the Block Data Rate of the VXI interface hardware, but rather by the total time required to both move and handle the data, or by the rate at which the instruments themselves can generate or accept the data.

Block Data Rate is easy for vendors to specify, but often difficult for users to relate to overall system performance. It is only one of many elements that affect the actual throughput of your system. For example, Block Data Rate does not indicate the processing power of your computer or the performance of the instruments themselves. In addition, a benchmark for Block Data Rate does not measure how fast you can control instruments using VXI Word Serial Protocol or random VXI reads and writes. The speed for Word Serial communication and random VXI reads and writes is dependent on the speed of the processor and the particular VXI instruments.

Local Performance

The MXIbus does not degrade the performance of the devices connected to it. Each MXI device can operate internally at full speed in parallel with other MXI devices. Because MXIbus is a true system bus with multimaster arbitration, the only time MXI devices must synchronize their operation is when they perform transactions that map across the MXIbus. When one MXI device performs a read or write that maps to a remote MXI device, the MXI hardware on both devices interlocks the bus cycle across the MXIbus to accomplish the transfer.

MXI – An Open Standard

The MXIbus specification was developed by National Instruments and announced in April 1989 as an open industry standard. A VXI*plug&play* core technology, MXIbus has been endorsed by the entire VXI*plug&play* Systems Alliance, including Tektronix, Hewlett-Packard, Racal Instruments, and GenRad. Because MXI is an open standard documented with a comprehensive specification, anyone can develop products that will be integrated into a MXI controlled system.

340243B-01
April 1996

3. PXI

What is PXI – PXI tutorial

Från National Instruments

Introduction

PXI (PCI eXtensions for Instrumentation) is a rugged PC-based platform for measurement and automation systems. PXI combines PCI electrical-bus features with the rugged, modular, Eurocard packaging of CompactPCI, and then adds specialized synchronization buses and key software features. PXI is both a high-performance and low-cost deployment platform for measurement and automation systems. These systems serve applications such as manufacturing test, military and aerospace, machine monitoring, automotive, and industrial test.

Developed in 1997 and launched in 1998, PXI was introduced as an open industry standard to meet the increasing demands of complex instrumentation systems. Today, PXI is governed by the PXI Systems Alliance (PXISA), a group of more than 65 companies chartered to promote the PXI standard, ensure interoperability, and maintain the PXI specification. For more information on the PXISA, including the PXI specification, refer to the PXISA website at www.pxisa.org.

Hardware Architecture

PXI systems are comprised of three basic components – chassis, system controller, and peripheral modules.

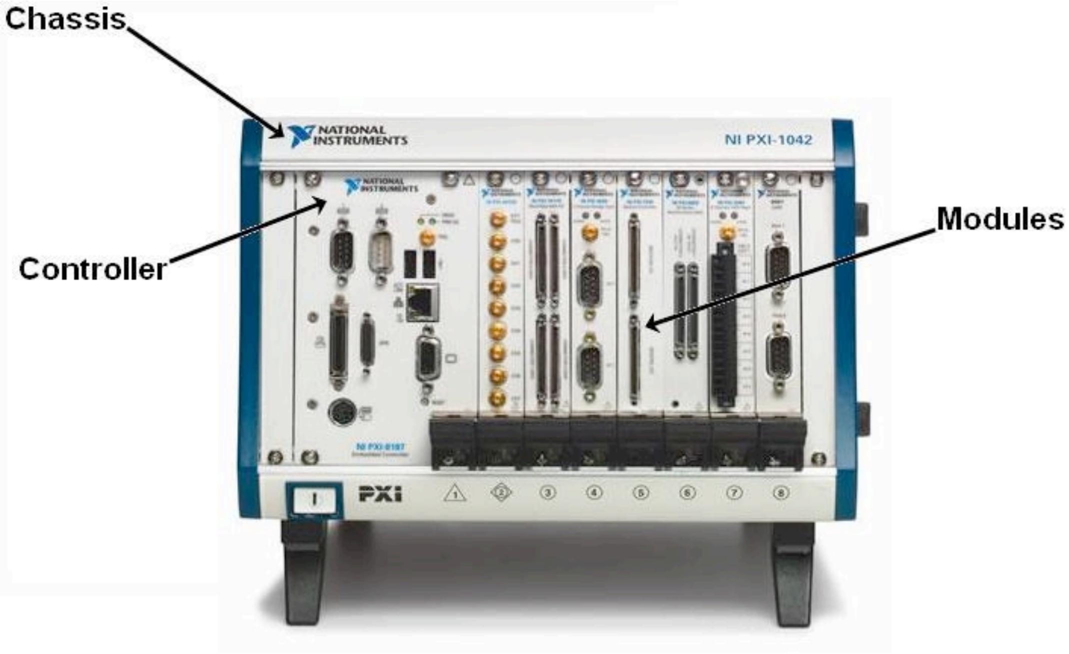


Figure 1. A standard 8-Slot PXI chassis contains an embedded system controller and seven peripheral modules.

PXI Chassis
The chassis provides the rugged and modular packaging for the system. Chassis, generally ranging in size from 4-slots to 18-slots, are also available with special features such as DC power supplies and integrated signal conditioning. The chassis contains the high-performance PXI backplane, which includes the PCI bus and timing and triggering buses (Figure 2). Using these timing and triggering buses, users can develop systems for applications requiring precise synchronization.

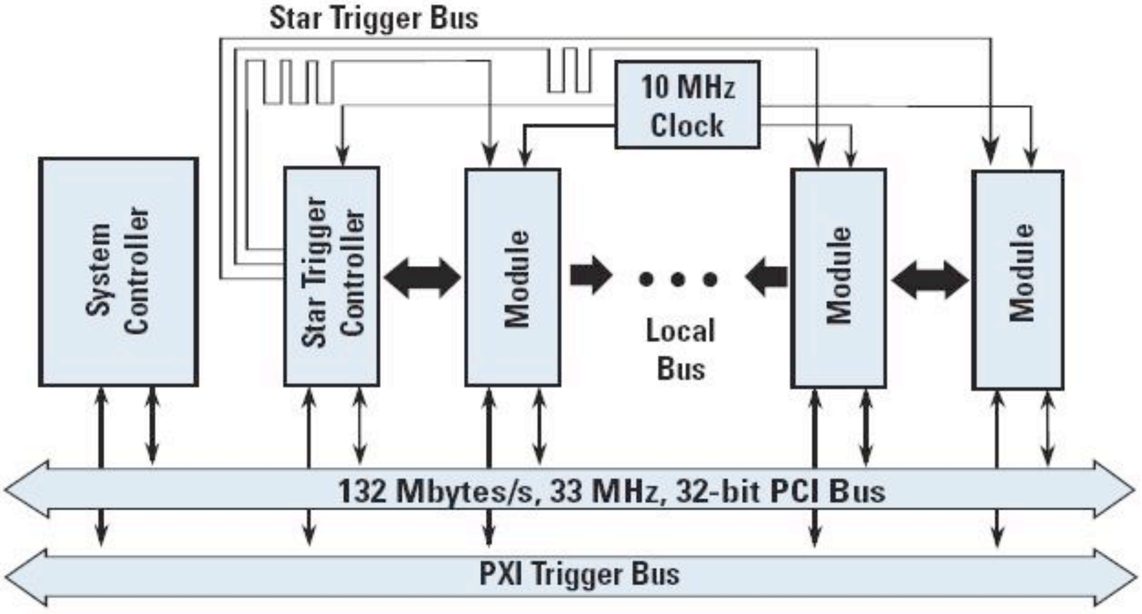


Figure 2. PXI Timing and Triggering Buses – PXI combines industry-standard PC components, such as the PCI bus, with advanced triggering and synchronization extensions on the backplane.

PXI Controllers
As defined by the PXI Hardware Specification, all PXI chassis contain a system controller slot located in the leftmost slot of the chassis (slot 1). Controller options include remote controllers from a desktop, workstation, server, or a laptop computer and high-performance embedded controllers with either a Microsoft OS (Windows 2000/XP) or a real-time OS (LabVIEW Real-Time)

<u>PXI Remote Controllers</u>
There are two types of PXI remote controllers:
* Laptop control of PXI
* PC control of PXI

Laptop Control of PXI
With ExpressCard MXI (Measurement eXtensions for Instrumentation) and PCMCIA CardBus interface kits, users can control PXI systems directly from laptop computers. During boot-up, the laptop computer will recognize all peripheral modules in the PXI system as PCI devices.

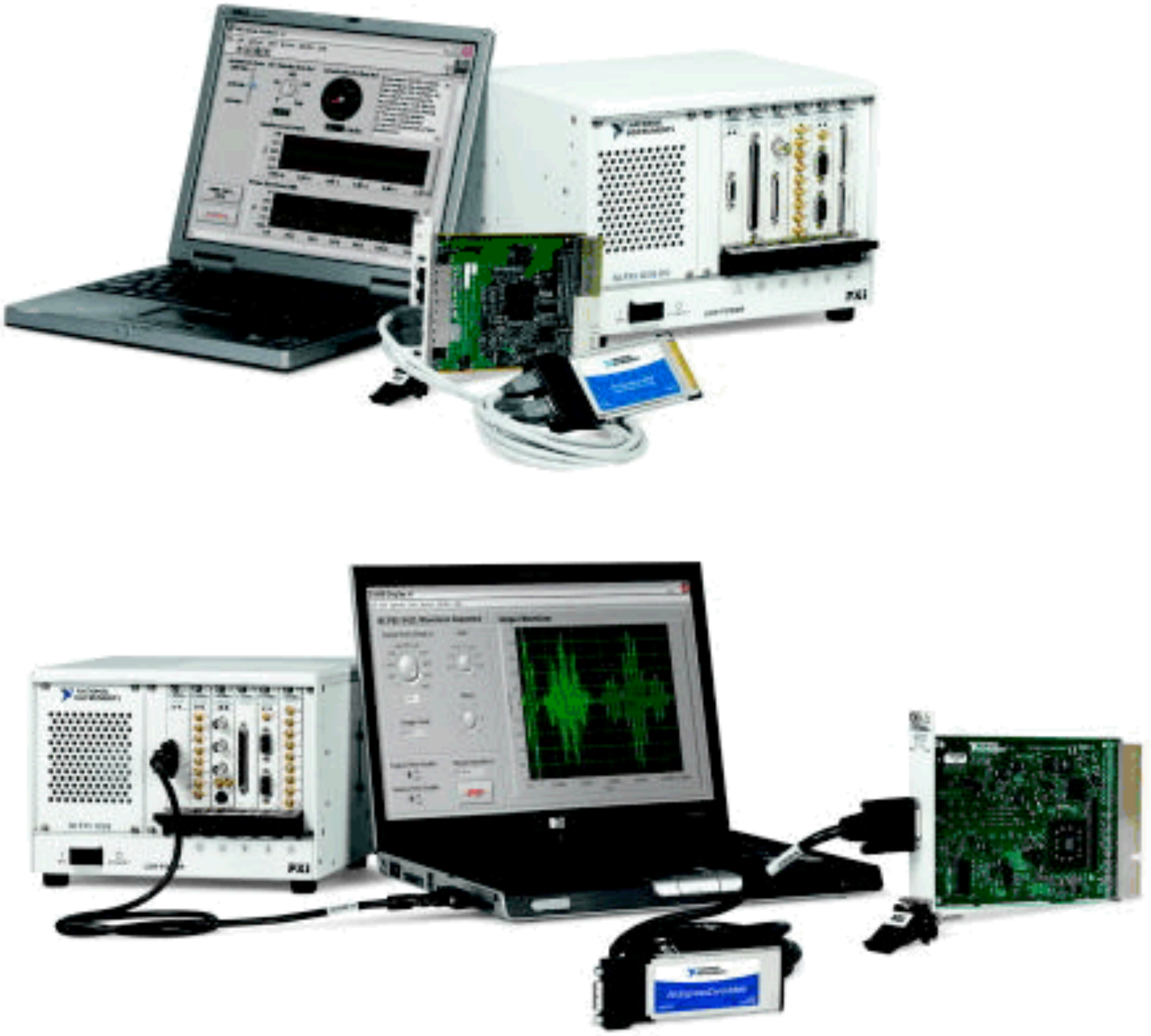


Figure 3. Laptop Control of PXI. PCMCIA CardBus interface kit (top), and ExpressCard MXI interface kit (bottom).

The ExpressCard MXI interface kit provides a 110 MB/s PCI Express -to- PCI bridge from the laptop computer to the PXI chassis. The PCMCIA CardBus interface kit provides a 50 MB/s PCI -to- PCI bridge from the laptop computer to the PXI chassis. Users now have the advantage of mobile/portable PXI systems with laptop control of PXI. You can purchase any ExpressCard MXI compatible laptop or PCMCIA CardBus compatible laptop to remotely control your PXI system.

PC Control of PXI

With MXI-Express and MXI-4 interface kits, users can control PXI systems directly from desktop, workstation, or server computers. During boot-up, the computer will recognize all peripheral modules in the PXI system as PCI devices.

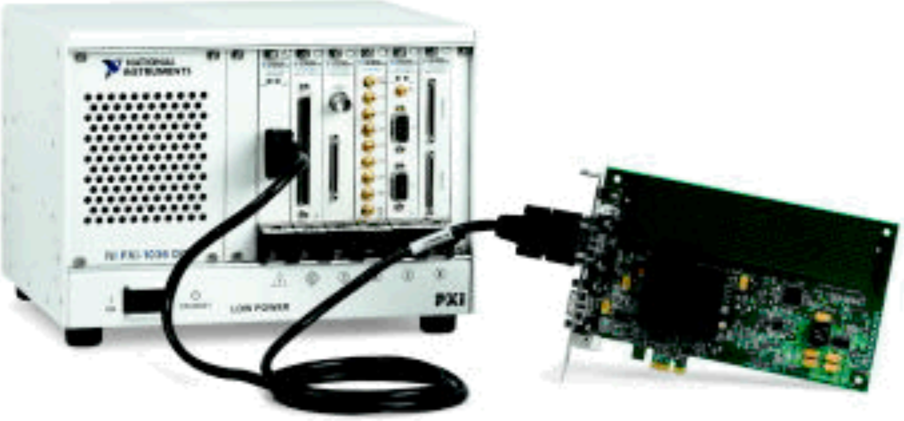


Figure 4a. Remote control with 2-port MXI-Express provides simultaneous control of two PXI chassis with combined 160 MB/s throughput.

The MXI-Express interface kit provides a 110 MB/s PCI Express -to- PCI bridge from the PC to the PXI chassis. With the NI PXI-PCIe8362 2-port interface kit, users will be able to control two PXI systems simultaneously from a single PC.

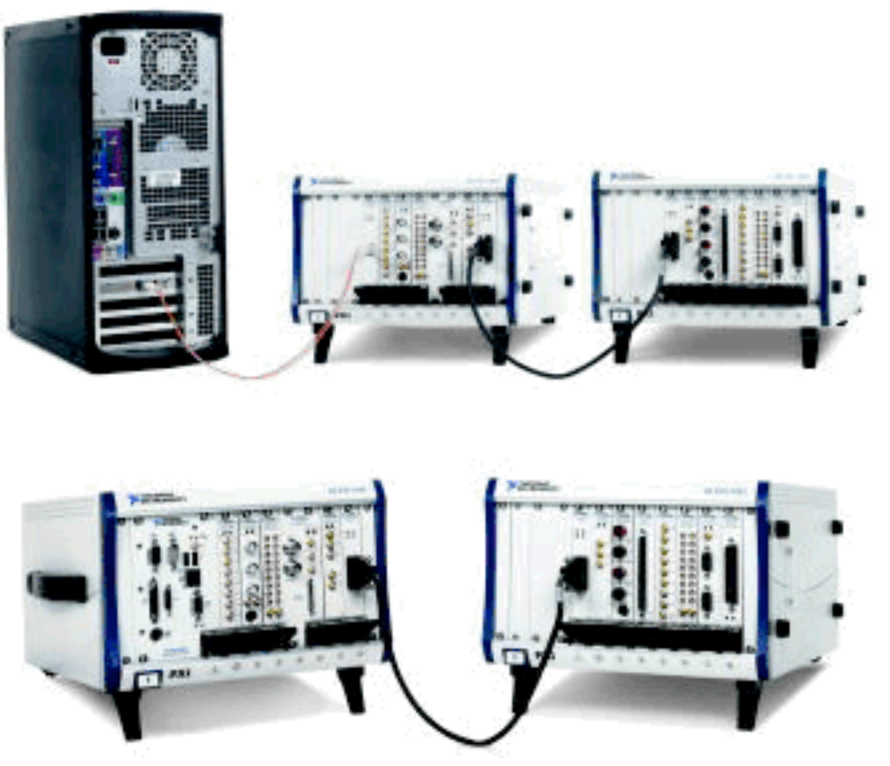


Figure 4b. Remote control with MXI-4 provides PC control of PXI, as well as multichassis PXI systems.

The MXI-4 interface kit provides a 78 MB/s PCI -to- PCI bridge from PC to the PXI system. MXI-4 interface kit comes with low-cost copper links or fiber-optic links for both extended distances and electrical isolation. As shown in Figure 4b, you can build multichassis PXI systems with MXI-4 as well. Using a MXI-4 link, you can implement either a daisy-chain or a star topology to build multichassis systems.

For more information on topologies for multichassis configurations, refer to the MXI-4 Series User Manual. You can purchase any desktop, workstation or server computer, and then remotely control your PXI system using either MXI-Express or copper/fiber optic MXI-4 serial link. For more information please refer to PC control of PXI.

With PXI remote controllers, you can maximize processor performance with minimized cost by using a desktop computer or laptop to remotely control a PXI system. Because all remote control products are software transparent, no additional programming is required.

PXI Embedded Controllers

Embedded controllers eliminate the need for an external PC, therefore providing a complete system contained in the PXI chassis. PXI embedded controllers are typically built using standard PC components in a small, PXI package. For example, the NI PXI-8187 controller has a Pentium 4-M 2.5 GHz processor, up to 1 GB of DDR RAM, a hard drive, and standard PC peripherals such as USB 2.0, Ethernet, serial, and parallel ports. Additionally, you can install your choice of OSs on the PXI controller, including Windows 2000/XP or LabVIEW Real-Time.

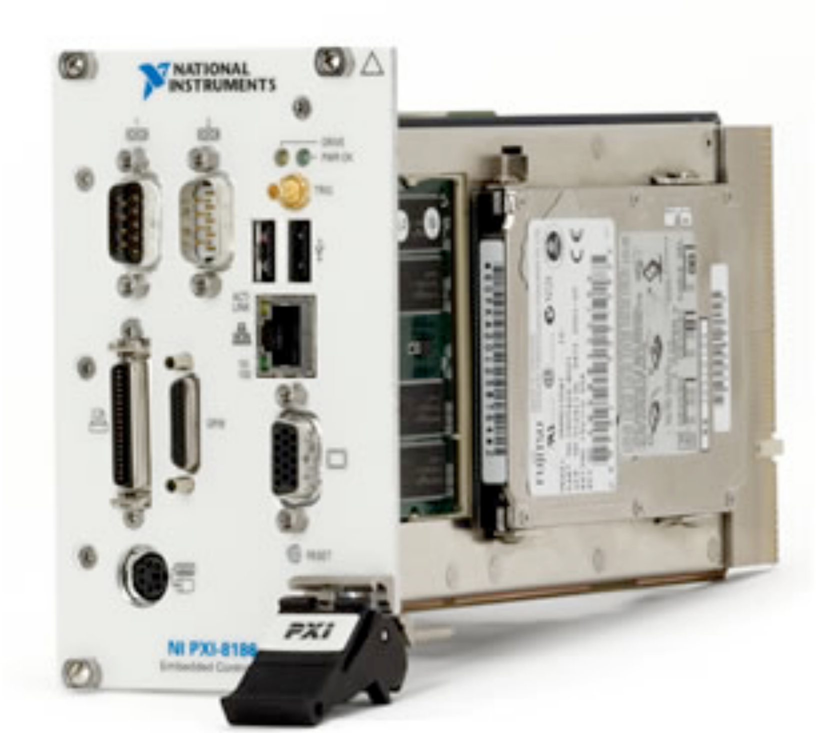
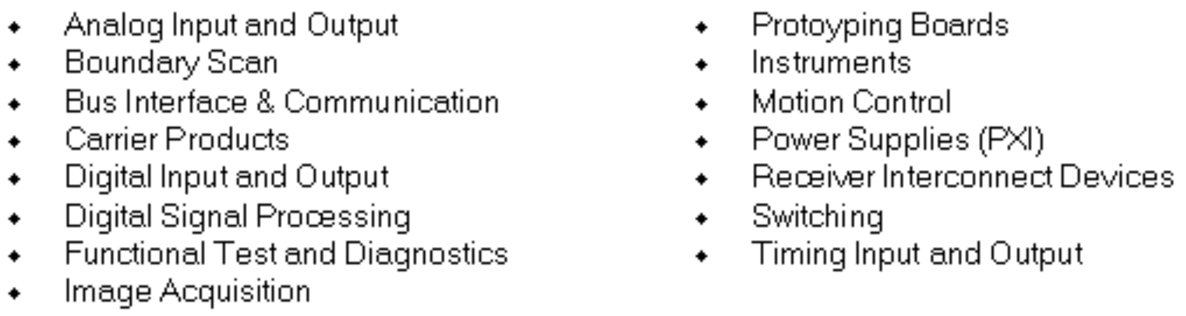


Figure 5. National Instruments PXI-8187 Pentium 4-M 2.5 GHz Embedded Controller. Notice the familiar PC peripherals such as keyboard/mouse and monitor connections, as well as the hard drive, USB 2.0, Ethernet, serial, and other standard PC peripherals. This controller runs standard Windows 2000/XP OSs, or can be targeted with LabVIEW Real-Time.

Embedded controllers are ideal for portable systems and contained "single-box" applications where the chassis is moved from one location to the next. For more information please refer to PXI controllers.

PXI Peripheral Modules

National Instruments offers more than 100 different PXI modules; and because PXI is an open industry standard, nearly 1000 modules are available from the 65+ members of the PXI Systems Alliance.



- Analog Input and Output
- Boundary Scan
- Bus Interface & Communication
- Carrier Products
- Digital Input and Output
- Digital Signal Processing
- Functional Test and Diagnostics
- Image Acquisition
- Protoyping Boards
- Instruments
- Motion Control
- Power Supplies (PXI)
- Receiver Interconnect Devices
- Switching
- Timing Input and Output

Because PXI is directly compatible with CompactPCI, you can use any 3U CompactPCI module in a PXI system. Additionally, CardBus/PCMCIA and PMC (PCI Mezzanine Card) cards can be installed in PXI systems using carrier modules. For example, the National Instruments PXI-8221 PC Card Carrier can be used to connect CardBus and PCMCIA devices to PXI systems.

PXI also preserves investments in stand-alone instruments or VXI systems by providing standard hardware and software for communication to these systems. For example, interfacing a PXI system to GPIB-based instrumentation is no different with a PXI-GPIB module than it is with a PCI-GPIB module. The software is identical. Additionally, a number of methods are available for interfacing PXI and VXI together. For more information, refer to the Web Event on Hybrid PXI and VXI Systems.

Software Architecture

Because PXI hardware is based on standard PC technologies, such as the PCI bus, as well as standard CPUs and peripherals, the standard Windows software architecture is familiar to users as well. Development and operation of Windows-based PXI systems is no different from that of a standard Windows-based PC. Additionally, because the PXI backplane uses the industry-standard PCI bus, writing software to communicate with PXI devices is, in most cases, identical to that of PCI devices. For example, software to communicate to an NI PXI-6251 multifunction data acquisition module is identical to that of an NI PCI-6251 board in a PC. Therefore, existing application software, example code, and programming techniques do not have to be rewritten or reused when moving software between PC-based and PXI-based systems.

As an alternative to Windows-based systems, you can use a real-time software architecture for time-critical applications requiring deterministic loop rates and headless operation (no keyboard, mouse, or monitor).

System Configuration

The fastest and easiest way to specify and configure your new PXI system is by using the online PXI Advisor or PXI/SCXI Advisor. The advisors lead you through a series of questions to help you build your new PXI system with a system controller, software, modules, accessories, and PXI or PXI/SCXI combination chassis. You build your system by answering simple questions and selecting the products best suited to your needs, and you can print or export the image of your PXI system for use in proposals or design reviews. Additionally, the advisors will make recommendations on technical matters, such as specific slot placement of modules, cables and terminal accessories, and integrated software packages. The advisors also use behind-the-scenes logic to prevent incompatible configurations. For example, if you select a LabVIEW Real-Time PXI controller, the advisors will automatically restrict PXI measurement module selection to only those compaatible with LabVIEW Real-Time.

When you are satisfied with your configuration, you can pass that configuration to a National Instruments representative for order, or you can automatically order through the online store. With NI Factory Installation Services as part of your order, you will receive your PXI system just as you configured it. NI installs your selected PXI modules in your chassis, and installs any memory upgrades, National Instruments application software, and required driver software on your embedded controller.

Summary

PXI modular instrumentation defines a rugged computing platform for measurement and automation users that clearly takes advantage of the technology advancements of the mainstream PC industry. By using the standard PCI bus, PXI modular instrumentation systems can benefit from widely available software and hardware components. The software applications and OSs that run on PXI systems are already familiar to users because they are already in use on common desktop computers. PXI meets your needs by adding rugged industrial packaging, plentiful slots for I/O, and features that provide advanced timing and triggering capabilities.

4. USB

An introduction to the USB-bus

Compiled from "A USB Primer" by Brian K. Lewis, Ph.D. Member of the Sarasota Personal Computer Users Group, Inc., "USB in a nutshell" by Craig Peacock, and "USB Complete: Everything You Need to Develop Custom USB Peripherals" by Jan Axelson

Introduction

The main reason for developing the Universal Serial Bus (USB) was to reduce the amount of cabling at the back of PCs. Apple developed the Apple Desktop Bus with this intention, where both the keyboard, mouse and some other peripherals could be connected together (daisy chained) using one cable, and this was the first development of the USB-bus.

USB ports have a number of advantages over the old system of parallel/serial ports. They do not require I/O memory space or individual IRQ lines, thus preventing IRQ conflicts when connecting external devices such as scanners or modems, a problem common to old computers. USB also allows for automatic device configuration and hot-plug capability. The hot-plug or hot-swap function means that you don't have to power down the computer and go through a restart when you want to connect a new device. In instead you simply connect or disconnect the USB cable. The computer will recognize the device and connect to the proper driver – if installed. Commonly, installation of drivers for external hard drives, printers, scanners, card readers, are usually necessary, while mice and keyboards that connect to the USB ports do not need specific drivers to be installed.

One of the benefits of USB is also bus-powered devices - devices which obtain its power from the bus and requires no external plug packs or additional cables.

The first USB standard was developed in the beginning of the nineties, commonly referred to as USB 1.0 or USB 1.1. By 2001 it was superseded by a standard that allowed for a much higher data transfer rate. USB 1 supported two speeds, a full speed mode of 12Mbits/s and a low speed mode of 1.5Mbits/s. The 1.5Mbits/s mode is slower and less susceptible to EMI, thus reducing the cost of ferrite beads and quality components. For example, crystals can be replaced by cheaper resonators. USB 2.0 allows a transfer rate of up to 480Mbits/s, known as High Speed mode, and was a tack on to compete with the Firewire Serial Bus. In conclusion there are three data transfer speeds defined within the USB standard:

- High Speed - 480Mbits/s
- Full Speed - 12Mbits/s
- Low Speed - 1.5Mbits/s

The Universal Serial Bus is host controlled, and there can only be one host per bus. The specification in itself does not support any form of multi-master arrangement. However the On-The-Go specification which is a tack on standard to USB 2.0 has introduced a Host Negotiation Protocol which allows two devices negotiate for the role of host. This is aimed at and limited to single point to point connections such as a mobile phone and personal organiser and not multiple hub, multiple device desktop configurations. The USB host is responsible for undertaking all transactions and scheduling bandwidth. Data can be sent by various transaction methods using a token-based protocol.

USB uses a tiered star topology, similar to that of 10BaseT Ethernet. This imposes the use of a hub somewhere, which adds to greater expense, more boxes on your desktop and more cables. However it is not as bad as it may seem, as many devices have USB hubs integrated into them. For example, your keyboard may contain a hub, which is connected to your computer. Your mouse and other devices such as your digital camera can be plugged easily into the back of your keyboard. Monitors are just another peripheral on a long list, which commonly have built-in hubs. Up to 127 devices can be connected to any one USB bus at any one given time.

The loading of the appropriate driver when a device in plugged into the bus, is done using a PID/VID (Product ID/Vendor ID) combination. The VID is supplied by the USB Implementor's forum at a cost and this is seen as another sticking point for USB. The latest info on fees can be found on the USB Implementor's Website. Other standards organizations provide an extra VID for non-commercial activities such as teaching, research or fiddling (The Hobbyist). In these cases you may wish to use one assigned to your development system's manufacturer. For example most chip manufacturers will have a VID/PID combination you can use for your chips which is known not to exist as a commercial device. Other chip manufacturers can even sell you a PID to use with their VID for your commercial device.

Another more notable feature of USB, is its transfer modes. USB supports Control, Interrupt, Bulk and Isochronous transfers. While we will look at the other transfer modes later, Isochronous allows a device to reserve a defined about of bandwidth with guaranteed latency. This is ideal in Audio or Video applications where congestion may cause loss of data or frames to drop. Each transfer mode provides the designer tradeoffs in areas such as error detection and recovery, guaranteed latency and bandwidth.

Hardware

Connectors

All devices have an upstream connection to the host and all hosts have a downstream connection to the device. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loopback connections at hubs such as a downstream port connected to a downstream port. There are commonly two types of connectors, called type A and type B which are shown below.

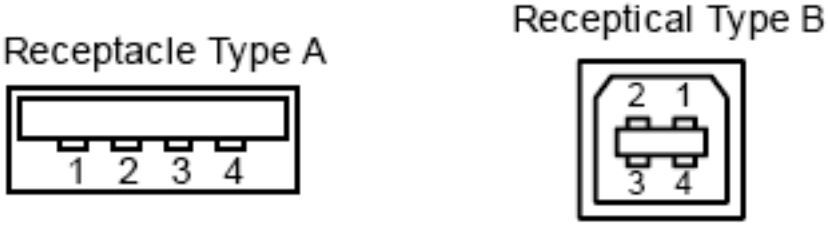
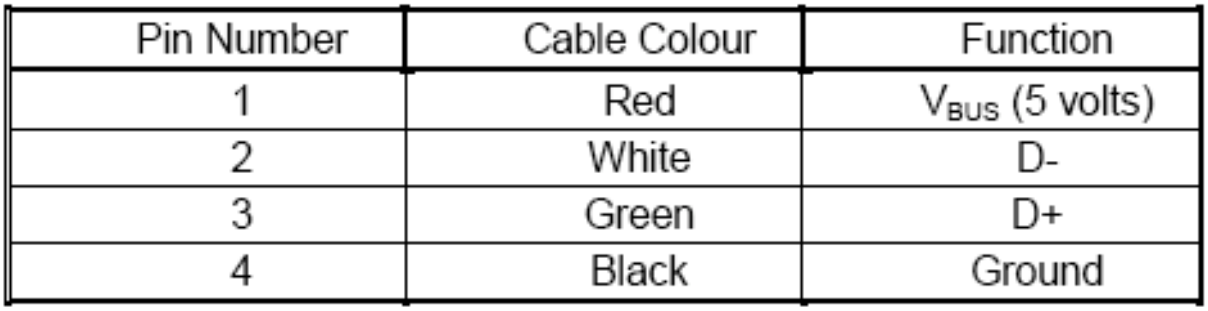


Figure 1: USB connectors

Type A plugs always face upstream. Type A sockets will typically find themselves on hosts and hubs. For example type A sockets are common on computer main boards and hubs. Type B plugs are always connected downstream and consequently type B sockets are found on devices. It is interesting to find type A to type A cables wired straight through and an array of USB gender changers in some computer stores. This is in contradiction of the USB specification. The only type A plug to type A plug devices are bridges which are used to connect two computers together. Other prohibited cables are USB extensions which has a plug on one end (either type A or type B) and a socket on the other. These cables violate the cable length requirements of USB. USB 2.0 included errata which introduces mini-USB B connectors. The details on these connectors can be found in Mini-B Connector Engineering Change Notice. The reasoning behind the mini connectors came from the range of miniature electronic devices such as mobile phones and organisers. The current type B connector is too large to be easily integrated into these devices. Just recently released has been the On-The-Go specification which adds peer-to-peer functionality to USB. This introduces USB hosts into mobile phone and electronic organisers, and thus has included a specification for mini-A plugs, mini-A receptacles, and mini-AB receptacles.



Pin Number	Cable Colour	Function
1	Red	V_{BUS} (5 volts)
2	White	D-
3	Green	D+
4	Black	Ground

Table 1 : USB Pin Functions

Standard internal wire colours are used in USB cables, making it easier to identify wires from manufacturer to manufacturer. The standard specifies various electrical parameters for the cables. It is interesting to read the detail the original USB 1.0 spec included. You would understand it specifying electrical attributes, but paragraph 6.3.1.2 suggested the recommended colour for overmolds on USB cables should be frost white - how boring! USB 1.1 and USB 2.0 was relaxed to recommend Black, Grey or Natural.

Electrical

Unless you are designing the silicon for a USB device/transceiver or USB host/hub, there is not all that much you need to know about the electrical specifications – the essential points will be addressed here. It uses four shielded wires of which two are power (+5V and GND). The remaining two are twisted pair differential data signals. It uses a NRZI (Non Return to Zero Invert) encoding scheme to send data with a sync field to synchronise the host and receiver clocks and is bit stuffed to ensure adequate transitions in the data stream.

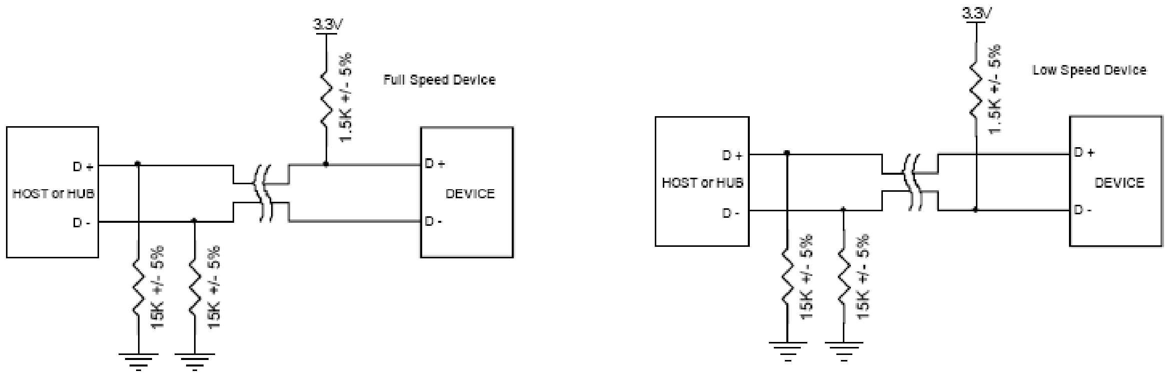


Figure 2: Full Speed Device with pull up resistor connected to D- (left). Low Speed Device with pull up resistor connected to D+ (right).

On low and full speed devices, a differential ‘1’ is transmitted by pulling D+ over 2.8V with a 15K ohm resistor pulled to ground and D- under 0.3V with a 1.5K ohm resistor pulled to 3.6V (see Fig 2). A differential ‘0’ on the other hand is a D- greater than 2.8V and a D+ less than 0.3V with the same appropriate pull down/up resistors. The receiver defines a differential ‘1’ as D+ 200mV greater than D- and a differential ‘0’ as D+ 200mV less than D-. The polarity of the signal is inverted depending on the speed of the bus. Therefore the terms ‘J’ and ‘K’ states are used in signifying the logic levels. In low speed a 'J' state is a differential 0. In high speed a 'J' state is a differential 1. USB transceivers will have both differential and single ended outputs. Certain bus states are indicated by single ended signals on D+, D- or both. For example a single ended zero or SE0 can be used to signify a device reset if held for more than 10mS. A SE0 is generated by holding both D- and D+ low (< 0.3V). Single ended and differential outputs are important to note if you are using a transceiver and FPGA as your USB device. You cannot get away with sampling just the differential output. The low speed/full speed bus has a characteristic impedance of 90 ohms +/- 15%. It is therefore important to observe the datasheet when selecting impedance matching series resistors for D+ and D-. Any good datasheet should specify these values and tolerances. High Speed (480Mbits/s) mode uses a 17.78mA constant current for signalling to reduce noise.

Speed Identification

A USB device must indicate its speed by pulling either the D+ or D- line high to 3.3 volts. A full speed device, pictured below will use a pull up resistor attached to D+ to specify itself as a full speed device. These pull up resistors at the device end will also be used by the host or hub to detect the presence of a device connected to its port. Without a pull up resistor, USB assumes there is nothing connected to the bus. Some devices have this resistor built into its silicon, which can be turned on and off under firmware control, others require an external resistor. For example Philips Semiconductor has a SoftConnect™ technology. When first connected to the bus, this allows the microcontroller to initialise the USB function device before it enables the pull up speed identification resistor, indicating a device is attached to the bus. If the pull up resistor was connected to Vbus, then this would indicate a device has been connected to the bus as soon as the plug is inserted. The host may then attempt to reset the device and ask for a descriptor when the microprocessor hasn't even started to initialise the usb function device. Other vendors such as Cypress Semiconductor also use a programmable resistor for Re-Numeration™ purposes in their EzUSB devices where the one device can be enumerated for one function such as In field programming then be disconnected from the bus under firmware control, and enumerate as another different device, all without the user lifting an eyelid. Many of the EzUSB devices do not have any Flash or OTP ROM to store code. They are bootstraped at connection. You will notice we have not included speed identification for High Speed mode. High speed devices will start by connecting as a full speed device (1.5k to 3.3V). Once it has been attached, it will do a high speed chirp during reset and establish a high speed connection if the hub supports it. If the device operates in high speed mode, then the pull up resistor is removed to balance the line. A USB 2.0 compliant device is not required to support high-speed mode. This allows cheaper devices to be produced if the speed isn't critical. This is also the case for a low speed USB 1.1 devices which is not required to support full speed. However a high speed device must not support low speed mode. It should only support full speed mode needed to connect first, then high speed mode if successfully negotiated later. An USB 2.0 compliant downstream facing device (Hub or Host) must support all three modes, high speed, full speed and low speed.

USB Protocols

Unlike RS-232 or similar serial interfaces where the format of data being sent is not defined, USB is made up of several layers of protocols. While this sounds complicated, don't give up now. Once you understand what is going on, you really only have to worry about the higher level layers. In fact most USB controller I.C.s will take care of the lower layer, thus making it almost invisible to the end designer. Each USB transaction consists of a

- Token Packet (Header defining what it expects to follow), an
- Optional Data Packet, (Containing the payload) and a

- Status Packet (Used to acknowledge transactions and to provide a means of error correction)

As we have already discussed, USB is a host centric bus. The host initiates all transactions. When the host powers up it polls each of the slave devices in turn (using the reserved address 0), it assigns each one a unique address and finds out from each device what its speed is and the and type of data transfer it wishes to perform. This process is called *enumeration* and it also takes place whenever a device is plugged into an active network. A typical transaction will consist of a number of packets - a token indicating the type of data that the host is sending or requiring, the data and in some cases an acknowledgement. Each packet is preceded by a sync field and followed by an end of packet marker.

Common USB Packet Fields

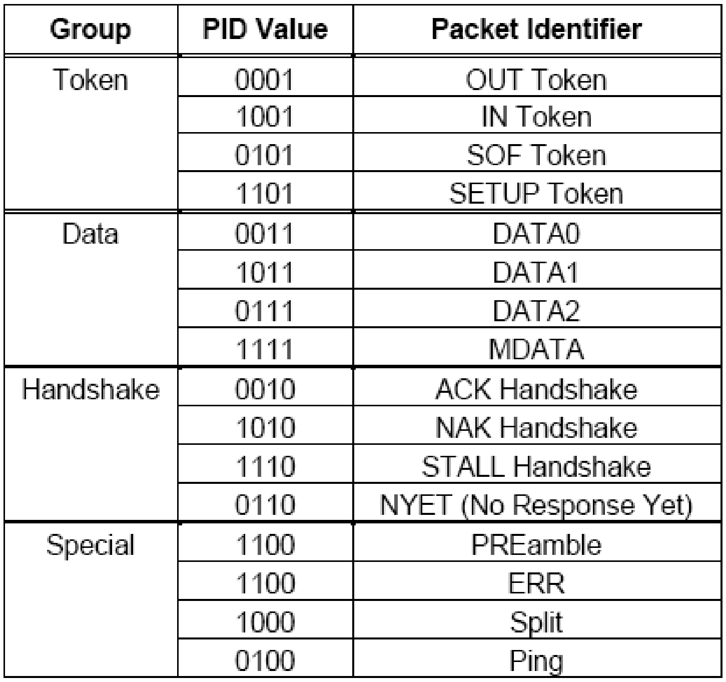
Data on the USBus is transmitted LSBit first. USB packets consist of the following fields,

- Sync

All packets must start with a sync field. The sync field is 8 bits long, which is used to synchronise the clock of the receiver with the transmitter. The last two bits indicate where the PID fields starts.

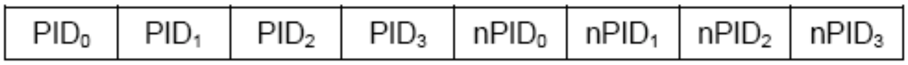
- PID

PID stands for Packet ID. This field is used to identify the type of packet that is being sent. The following table shows the possible values.



Group	PID Value	Packet Identifier
Token	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET (No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

There is 4 bits to the PID, however to insure it is received correctly, the 4 bits are complemented and repeated, making an 8 bit PID in total. The resulting format is shown below.



PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

- ADDR

The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. Address 0 is not valid, as any device which is not yet assigned an address must respond to packets sent to address zero.

- ENDP

The endpoint field is made up of 4 bits, allowing 16 possible endpoints. Low speed devices, however can only have two endpoint additional addresses on top of the default pipe. (four endpoints max).

- CRC

Cyclic Redundancy Checks are performed on the data within the packet payload. All token packets have a 5 bit CRC while data packets have a 16 bit CRC.

- EOP

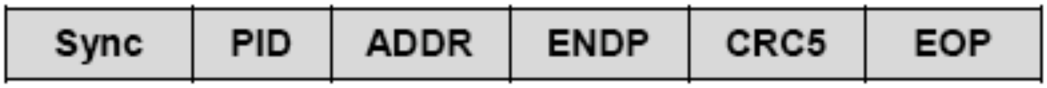
End of packet. Signalled by a Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time.

Common USB Packet Types

USB has four different packet types. Token packets indicate the type of transaction to follow, data packets contain the payload, handshake packets are used for acknowledging data or reporting errors and start of frame packets indicate the start of a new frame.

- Token Packets

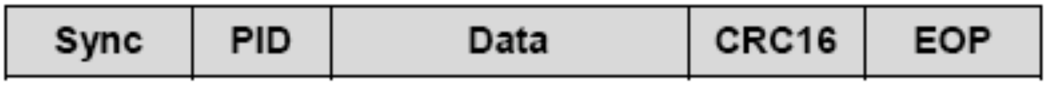
There are three types of token packets:
In – Informs the USB device that the host wishes to read information.
Out - Informs the USB device that the host wishes to send information.
Setup – Used to begin control transfers.
Token Packets must conform to the following format:



Sync	PID	ADDR	ENDP	CRC5	EOP

- Data Packets

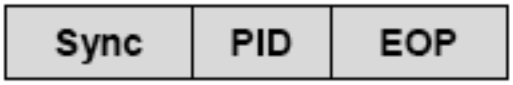
There are two types of data packets each capable of transmitting 0 to 1023 bytes of data, called Data0 and Data1. Data packets have the following format:



Sync	PID	Data	CRC16	EOP

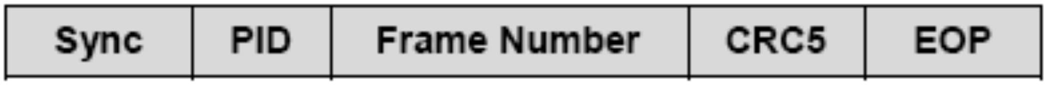
- **Handshake Packets** There are three type of handshake packets which consist simply of the PID ACK – Acknowledgment that the packet has been successfully received.

NAK – Reports that the device cannot send nor received data temporary. Also used during interrupt transaction to inform the host there is no data to send.
STALL – The device finds its in a state that it requires intervention from the host.
Handshake Packets have the following format:



Sync	PID	EOP

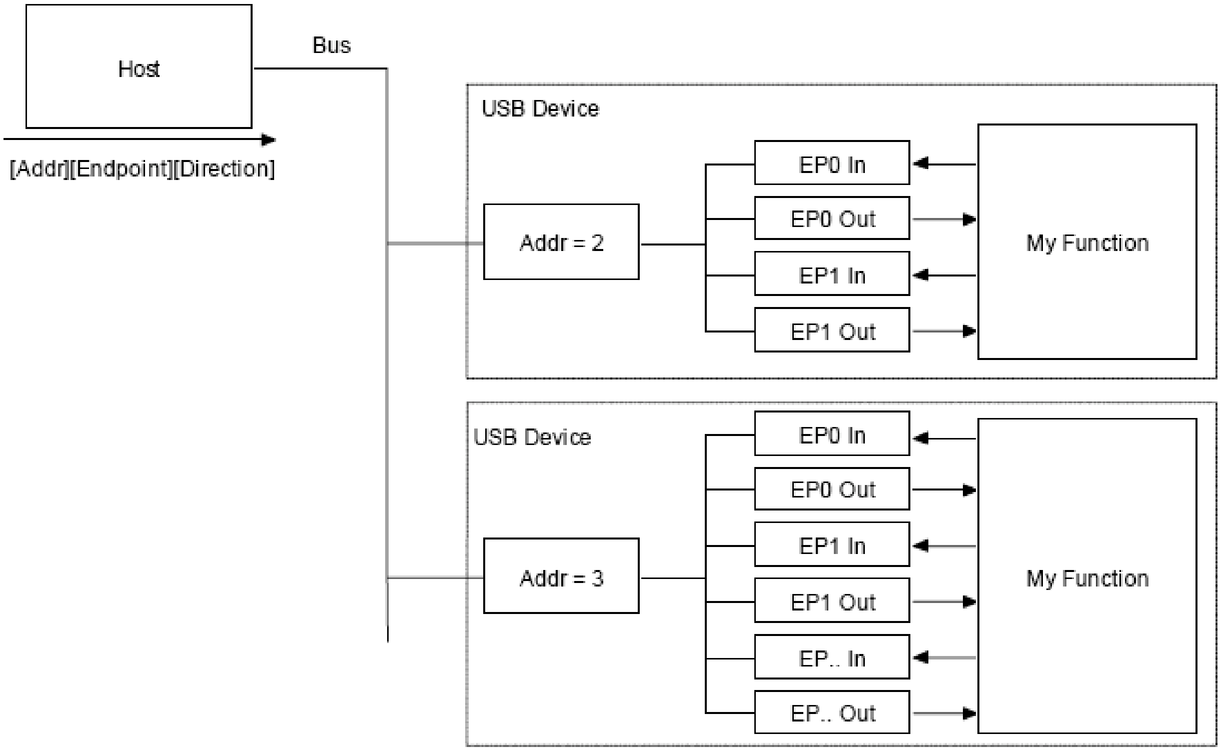
- **Start of Frame Packets** The SOF packet consisting of an 11-bit frame number is sent by the host every 1mS ± 500nS.



Sync	PID	Frame Number	CRC5	EOP

USB Functions

When we think of a USB device, we think of a USB peripheral, but a USB device could mean a USB transceiver device used at the host or peripheral, a USB Hub or Host Controller IC device, or a USB peripheral device. The standard therefore makes references to USB functions which can be seen as USB devices which provide a capability or function such as a Printer, Zip Drive, Scanner, Modem or other peripheral.



Most functions will have a series of buffers, typically 8 bytes long. Each buffer will belong to what is referred to as an endpoint - EP0 IN, EP0 OUT etc (see below). Say for example, the host sends a device descriptor request. The function hardware will read the setup packet and determine from the address field whether the packet is for itself, and if so will copy the payload of the following data packet to the appropriate endpoint buffer dictated by the value in the endpoint field of the setup token. It will then send a handshake packet to acknowledge the reception of the byte and generate an internal interrupt within the semiconductor/micro-controller for the appropriate endpoint signifying it has received a packet. This is typically all done in hardware. The software now gets an interrupt, and should read the contents of the endpoint buffer and parse the device descriptor request.

Endpoints

Endpoints can be described as sources or sinks of data. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. At the software layer, your device driver may send a packet to your devices EP1 for example. As the data is flowing out from the host, it will end up in the EP1 OUT buffer. Your firmware will then at its leisure read this data. If it wants to return data, the function cannot simply write to the bus as the bus is controlled by the host. Therefore it writes data to EP1 IN which sits in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen as the interface between the hardware of the function device and the firmware running on the function device. All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

Pipes

While the device sends and receives data on a series of endpoints, the client software transfers data through pipes. A pipe is a logical connection between the host and endpoint(s). Pipes will also have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type (Control, Bulk, Iso or Interrupt) it uses, a direction of data flow and maximum packet/buffer sizes. For example the default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type. USB defines two types of pipes

- Stream Pipes have no defined USB format, that is you can send any type of data down a stream pipe and can retrieve the data out the other end. Data flows sequentially and has a pre-defined direction, either in or out. Stream pipes will support bulk, isochronous and interrupt transfer types. Stream pipes can either be controlled by the host or device.
- Message Pipes have a defined USB format. They are host controlled, which are initiated by a request sent from the host. Data is then transferred in the desired direction, dictated by the request. Therefore message pipes allow data to flow in both directions but will only support control transfers.

Endpoint types

The Universal Serial Bus specification defines four transfer/endpoint types:

- Control Transfers

Control transfers are typically used for command and status operations. They are essential to set up a USB device with all enumeration functions being performed using control transfers. They are typically bursty, random packets which are initiated by the host and use best effort delivery. The packet length of control transfers in low speed devices must be 8 bytes, high speed devices allow a packet size of 8, 16, 32 or 64 bytes and full speed devices must have a packet size of 64 bytes.

- Interrupt Transfers

Any one who has had experience of interrupt requests on microcontrollers will know that interrupts are device generated. However under USB if a device requires the attention of the host, it must wait until the host polls it before it can report that it needs urgent attention! Interrupt Transfers are unidirectional and use a stream pipe. This type of transfers are typically non-periodic, small device "initiated" communication requiring bounded latency, such as a mouse or keyboard. An Interrupt request is queued by the device until the host polls the USB device asking for data. The maximum data payload size for low-speed devices is 8 bytes, for full-speed devices is 64 bytes, and for high-speed devices 1024 bytes.

• Isochronous Transfers

Isochronous transfers occur continuously and periodically. They typically contain time sensitive information, such as an audio or video stream. If there were a delay or retry of data in an audio stream, then you would expect some erratic audio containing glitches. The beat may no longer be in sync. However if a packet or frame was dropped every now and again, it is less likely to be noticed by the listener. Isochronous Transfers provide

- Guaranteed access to USB bandwidth.
- Bounded latency.
- Stream Pipe - Unidirectional
- Error detection via CRC, but no retry or guarantee of delivery.
- Full & high speed modes only.
- No data toggling.

The maximum size data payload is specified in the endpoint descriptor of an Isochronous Endpoint. This can be up to a maximum of 1023 bytes for a full speed device and 1024 bytes for a high speed device. As the maximum data payload size is going to effect the bandwidth requirements of the bus, it is wise to specify a conservative payload size. If you are using a large payload, it may also be to your advantage to specify a series of alternative interfaces with varying isochronous payload sizes. If during enumeration, the host cannot enable your preferred isochronous endpoint due to bandwidth restrictions, it has something to fall back on rather than just failing completely. Data being sent on an isochronous endpoint can be less than the prenegotiated size and may vary in length from transaction to transaction. Isochronous transactions do not have a handshaking stage and cannot report errors or STALL/HALT conditions.

• Bulk Transfers

Bulk transfers can be used for large bursty data. Such examples could include a print-job sent to a printer or an image generated from a scanner. Bulk transfers provide error correction in the form of a CRC16 field on the data payload and error detection/re-transmission mechanisms ensuring data is transmitted and received without error. Bulk transfers will use spare un-allocated bandwidth on the bus after all other transactions have been allocated. If the bus is busy with isochronous and/or interrupt then bulk data may slowly trickle over the bus. As a result Bulk transfers should only be used for time insensitive communication as there is no guarantee of latency. Bulk Transfers

- are used to transfer large bursty data.
- have error detection via CRC, with guarantee of delivery.
- No guarantee of bandwidth or minimum latency.
- Stream Pipe - Unidirectional
- Full and high speed modes only.

Bulk transfers are only supported by full and high speed devices. For full speed endpoints, the maximum bulk packet size is either 8, 16, 32 or 64 bytes long. For high speed endpoints, the maximum packet size can be up to 512 bytes long. If the data payload falls short of the maximum packet size, it does not need to be padded with zeros. A bulk transfer is considered complete when it has transferred the exact amount of data requested, transferred a packet less than the maximum endpoint size of transferred a zero-length packet.

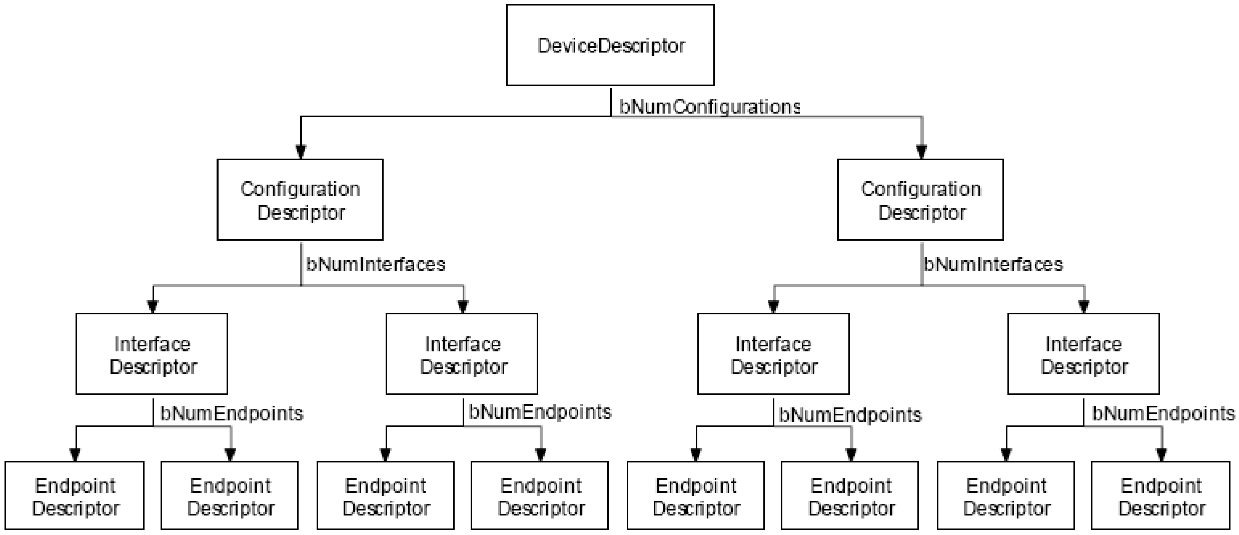
USB Descriptors

All USB devices have a hierarchy of descriptors which describe to the host information such as what the device is, who makes it, what version of USB it supports, how many ways it can be configured, the number of endpoints and their types etc. The more common USB descriptors are

- Device Descriptors
- Configuration Descriptors
- Interface Descriptors
- Endpoint Descriptors
- String Descriptors

USB devices can only have one device descriptor. The device descriptor includes information such as what USB revision the device complies to, the Product and Vendor IDs used to load the appropriate drivers and the number of possible configurations the device can have. The number of configurations indicate how many configuration descriptors branches are to follow.

The configuration descriptor specifies values such as the amount of power this particular configuration uses, if the device is self or bus powered and the number of interfaces it has. When a device is enumerated, the host reads the device descriptors and can make a decision of which configuration to enable. It can only enable one configuration at a time. For example, it is possible to have a high power bus powered configuration and a self powered configuration. If the device is plugged into a host with a mains power supply, the device driver may choose to enable the high power bus powered configuration enabling the device to be powered without a connection to the mains, yet if it is connected to a laptop or personal organiser it could enable the second configuration (self powered) requiring the user to plug your device into the power point. The configuration settings are not limited to power differences. Each configuration could be powered in the same way and draw the same current, yet have different interface or endpoint combinations. However it should be noted that changing the configuration requires all activity on each endpoint to stop. While USB offers this flexibility, very few devices have more than one configuration.



The interface descriptor could be seen as a header or grouping of the endpoints into a functional group performing a single feature of the device. For example you could have a multi-function fax/scanner/printer device. Interface descriptor one could describe the endpoints of the fax function, Interface descriptor two the scanner function and Interface descriptor three the printer function. Unlike the configuration descriptor, there is no limitation as to having only one interface enabled at a time. A device could have one or many interface descriptors enabled at once. Interface descriptors have a **bInterfaceNumber** field specifying the Interface number and a **bAlternateSetting** which allows an interface to change settings on the fly. For example we could have a device with two interfaces, interface one and interface two. Interface one has **bInterfaceNumber** set to zero indicating it is the first interface descriptor and a **bAlternativeSetting** of zero. Interface two would have a **bInterfaceNumber** set to one indicating it is the second interface and a **bAlternativeSetting** of zero (default). We could then throw in another descriptor, also with a **bInterfaceNumber** set to one indicating it is the second interface, but this time setting the bAlternativeSetting to one, indicating this interface descriptor can be an alternative setting to that of the other interface descriptor two. When this configuration is enabled, the first two interface descriptors with bAlternativeSettings equal to zero is used. However during operation the host can send a SetInterface request directed to that of Interface one with an alternative setting of one to enable the other interface descriptor. This gives an advantage over having two configurations, in that we can be transmitting data over interface zero while we change the endpoint settings associated with interface one without effecting interface zero.

Each endpoint descriptor is used to specify the type of transfer, direction, polling interval and maximum packet size for each endpoint. Endpoint zero, the default control endpoint is always assumed to be a control endpoint and as such never has a descriptor.

String descriptors provide human readable information and are optional. If they are not used, any string index fields of descriptors must be set to zero indicating there is no string descriptor available.

Enumeration

Enumeration is the process of determining what device has just been connected to the bus and what parameters it requires such as power consumption, number and type of endpoint(s), class of product etc. The host will then assign the device an address and enable a configuration allowing the device to transfer data on the bus. A fairly generic enumeration process is detailed in section 9.1.2 of the USB specification. However when writing USB firmware for the first time, it is handy to know exactly how the host responds during enumeration, rather than the general enumeration process detailed in the specification. A common Windows enumeration involves the following steps:

1. The host or hub detects the connection of a new device via the device's pull up resistors on the
data pair. The host waits for at least 100ms allowing for the plug to be inserted fully and for power
to stabilise on the device.
2. Host issues a reset placing the device is the default state. The device may now respond to the default address zero.
3. The MS Windows host asks for the first 64 bytes of the Device Descriptor.
4. After receiving the first 8 bytes of the Device Descriptor, it immediately issues another bus reset.
5. The host now issues a Set Address command, placing the device in the addressed state.
6. The host asks for the entire 18 bytes of the Device Descriptor.
7. It then asks for 9 bytes of the Configuration Descriptor to determine the overall size.
8. The host asks for 255 bytes of the Configuration Descriptor.
9. Host asks for any String Descriptors if they were specified.

At the end of Step 9, Windows will ask for a driver for your device. It is then common to see it request all the descriptors again before it issues a Set Configuration request. The above enumeration process is common to Windows 2000, Windows XP and Windows 98 SE. Step 4 often confuses people writing firmware for the first time. The Host asks for the first 64 bytes of the device descriptor, so when the host resets your device after it receives the first 8 bytes, it is only natural to think there is something wrong with your device descriptor or how your firmware handles the request. However as many will tell you, if you keep persisting by implementing the Set Address Command it will pay off by asking for a full 18 bytes of device descriptor next. Normally when something is wrong with a descriptor or how it is being sent, the host will attempt to read it three times with long pauses in between requests. After the third attempt, the host gives up reporting an error with your device.

5. Fältbussar

Från: http://www.infoside.de/infida/wissen_allgemein.htm

What is Fieldbus?

Fieldbus is a generic-term which describes a new digital communications network which will be used in industry to replace the existing 4 - 20mA analogue signal. The network is a digital, bi-directional, multidrop, serial-bus, communications network used to link isolated field devices, such as controllers, transducers, actuators and sensors. Each field device has low cost computing power installed in it, making each device a 'smart' device. Each device will be able to execute simple functions on it's own such as diagnostic, control, and maintenance functions as well as providing bi-directional communication capabilities. With these devices not only will the engineer be able to access the field devices, but they are also able to communicate with other field devices. In essence fieldbus will replace centralised control networks with distributed control networks. Therefore fieldbus is much more than a replacement for the 4 - 20mA analogue standard.

The fieldbus technology promises to improve quality, reduce costs and boost efficiency. These promises made by the fieldbus technology are derived partly from the fact that information which a field device is required to transmit or receive can be transmitted digitally. This is a great deal more accurate than transmitting using analogue methods which were used previously. Each field device is also a 'smart' device and can carry out it's own control, maintenance and diagnostic functions. As a result it can report if there is a failure of the device or manual calibration is required, this increases the efficiency of the system and reduces the amount of maintenance required.

Each field device will be more flexible as they will have computing power. One fieldbus device could be used to replace a number of devices using the 4 - 20mA analogue standard. Other major cost savings from using fieldbus are due to wiring and installation - the existing 4 - 20mA analogue signal standard requires each device to have is own set of wires and its own connection point. Fieldbus eliminates this need so only a single twisted pair wiring scheme is required.

Fieldbus Organisations

This section establishes who have been the major instigators of fieldbus development over the past several years. A brief summary of the developing standard is also covered.

The major players in the fieldbus area were previously dominated by two major groups:

- WorldFIP (World Factory Instrumentation Protocol)
- ISP (Interoperable Systems Project)

However, recently, these two groups have joined together to form the Fieldbus Foundation (FF). The Fieldbus Foundation and another organisation known as Profibus-ISP are now competing for market dominance.

Two standards bodies known as the IEC (International Electrotechnical Commission) and the ISA (Industry Society of America) are currently working on an international standard known as SP50. This standard will hopefully allow the manufacturers of fieldbus equipment all around the world to produce compatible instruments for industrial applications. WorldFIP, ISP and FF have pledged that they will eventually evolve their products to meet the standard when it arrives. However, when the standard finally does arrive, users of existing non-conforming equipment will run the risk of having obsolete equipment or having to purchase new systems at an excessive cost.

At the time of writing, information regarding actual market share for the Fieldbus Foundation and Profibus-ISP was not available, but Process Engineering's Instrumentation Supplement for 1994, predicts that the Fieldbus Foundation will take the greater market share.

World Factory Instrumentation Protocol

The World Factory Instrumentation Protocol (WorldFIP) was developed from an earlier French National Standard known as NFC 46-600, or more commonly as FIP. It is a consortium of companies producing field bus instruments that use a messaging system.
Time critical options are supposedly guaranteed in a WorldFIP implementation. WorldFIP plans to add a device description tool, known as the WorldFIP Device Builder. The Device Builder will automatically inform the control system what features and parameters each instrument connected to the bus has.

Interestingly , WorldFIP is divisional in nature with a UK, European and North American division. Each division is motivated by similar goals and similar implementations, but each operates almost autonomously from the others.

Some of the major members of WorldFIP include:

Honeywell (Arizona)
Bailey Controls (Ohio)
Cegelec (Paris)
Allen Bradley Corporation (Ohio)
Telemecanique (Paris)
Ronan Engineering Co. (California)
Square D
Electricite de France (France)
Elf (France)

Interoperable Systems Project

The Interoperable Systems Project (ISP) implementation is based on the German National Standard DIN STD19245, also known as Process Field Bus, or Profibus. Profibus is similar to the token passing network commonly implemented on many networks today. The ISP extension to Profibus is the Device Description Language (DDL). DDL allows an instrument added to the bus system to communicate to a master control what its functions and capabilities are.

Some of the major members of ISP include:

Siemens (Germany)
The Rosemount Group (Minnesota)
Fisher Controls, Inc. (Texas)
Foxoboro Co. (Massachusetts)
ABB Co. (Sweden)
Yokogawa Electric Corporation (Tokyo)

Fieldbus Foundation

On a positive note ISP and WorldFIP (North American division) have been working together since late 1993 on a possible merger of their technology. A single solution has been what industry has needed for a long time, so in June of 1994, the Fieldbus Foundation (FF) was set up between ISP and WorldFIP (NA). However, at least 1 to 2 years of delay is expected before a complete product can be produced.

Profibus-ISP

Effectively a breakaway group of the Profibus and ISP organisations, this group effectively announced to the world that they will have their own fieldbus communications system ready in approximately June/July 1994. Profibus-ISP is derived from the Profibus and ISP products, and hence has the features of both with some small additions.

At the time of writing, little information on Profibus-ISP and the Fieldbus Foundation was available.

IEC/ISA SP50

The ISA/IEC are developing a standard with the working name of SP50. The standard will follow the ISO/OSI seven layer model for data communications with an additional eighth layer which focuses on the product interoperablility.

Current progress on the SP50 is as follows

Physical - Completed. Specification includes
31.25 kbit/sec, 1 Mbit/sec and 2.5 Mbit/sec data transfer rates.
Requirements for fieldbus component parts.
Media and network configuration requirements for data integrity and interoperability between devices.

Från: http://www.kvaser.se/can/general.htm

The CAN bus

The CAN bus is a broadcast type of bus. This means that all nodes can "hear" all transmissions. There is no way to send a message to just a specific node; all nodes will invariably pick up all traffic. The CAN hardware, however, provides local filtering so that each node may react only on the interesting messages.

The bus uses two different signaling states: dominant (logically 0) and recessive (logically 1). These correspond to certain electrical levels which depend on the physical layer used (there are several.) The modules are connected to the bus in a wired-and fashion: if just one node is driving the bus to the dominant state, then the whole bus is in that state regardless of the number of nodes transmitting a recessive state.

In order to eliminate the DC component on the bus, the CAN messages are bit-stuffed, that is, for every occurrence of five consecutive bits of the same level, the transmitter inserts an extra bit of the opposite level. The receiver removes these extra bits.

The CAN messages

CAN uses short messages - the maximum utility load is 94 bits. There is no explicit address in the messages; instead, the messages can be said to be contents-addressed, that is, their contents implicitly determines their address.

Message Types

There are four different message types (or "frames") on a CAN bus:

- the Data Frame,
- the Remote Frame,
- the Error Frame, and
- the Overload Frame.

The Data Frame

The Data Frame is the most common message type. It comprises the following important parts (a few details are omitted for the sake of brevity):

- the Arbitration Field, which determines the priority of the message when two or more nodes are contending for the bus. The contents of the Arbitration field, which is 11 or 29 bits long, is usually called the Identifier; however, there is no obligation that the Arbitration field really contains any kind of identifier!
- the Data Field, which contains zero to eight bytes of data.
- the CRC Field, which contains a 16-bit checksum calculated on most parts of the message. This checksum is used for error detection.
- an Acknowledgement Slot; any CAN controller that has been able to correctly receive the message sends an Acknowledgement bit at the end of each message. The transmitter checks

for the presence of the Acknowledge bit and retransmits the message if no acknowledge was detected.

It is worth noting that the presence of an Acknowledgement Bit on the bus does not mean that any of the intended addressees has received the message, just that one node on the bus has received it correctly.

The Remote Frame

The Remote Frame is just like the Data Frame, with two important differences:

- it is explicitly marked as a Remote Frame (a certain control bit is used for this purpose), and
- there is no Data Field.

The intended purpose of the Remote Frame is to solicit the transmission of the corresponding Data Frame. If, say, node A transmits a Remote Frame with the Arbitration Field set to 234, then node B, if properly initialized, might respond with a Data Frame with the Arbitration Field also set to 234. This can be used to implement a type of request-response type of bus traffic management. In practice, however, the Remote Frame is little used. It is also worth noting that the CAN standard does not prescribe the behaviour outlined here. Most CAN controllers can be programmed either to automatically respond to a Remote Frame, or to notify the local CPU instead.

Sometimes it is claimed that the node responding to the Remote Frame is starting its transmission as soon as the identifier is recognized, thereby "filling up" the empty Remote Frame. This is not the case.

The Error Frame

Simply put, the Error Frame is a special message that violates the rules of CAN. It is transmitted when a node detects a fault and will cause all other nodes to detect a fault. The transmitter can then retransmit the message. There is an elaborate scheme of error counters that ensures that a node can't destroy the bus traffic by repeatedly transmitting Error Frames.

The Overload Frame

The Overload Frame is mentioned here just for completeness. It is very similar to the Error Frame and is transmitted by a node that becomes too busy. The Overload Frame is not used very often, as today's CAN controllers are clever enough not to use it.

CAN Physical Layer(s)

A physical layer defines the electrical levels and signaling scheme on the bus, the cable impedance and similar things.

There are a few different physical layers:

- ISO 11898 defines a two-wire balanced signaling scheme.
- Little-used ISO 11519 for lower speed applications defines another two-wire balanced signaling scheme for lower bus speeds.
- Several proprietary physical layers do exist.

- A modified type of RS485 has been used.

Most CAN transceiver chips are manufactured by Philips; alternative vendors include Siliconix and Unitrode.

A very common type is the 82C250 transceiver which implements the physical layer defined by ISO 11898.

The ISO 11898 prescribes that the cable impedance be nominally 120 Ohms; an impedance in the interval of [108..132] Ohms is permitted.

There are not many cables in the market today that fulfils this requirement. There is a good chance that the allowed impedance interval will be broadened in the future; see CANHUG's proposal.

The maximum speed of a CAN bus, according to the standard, is 1 Mbit/second. At this speed, a maximum cable length of about 40 metres (130 ft.) can be used. This is because the arbitration scheme requires that the wave front can propagate to the most remote node and back again before the bit is sampled. In other words, the cable length is restricted by the speed of light. A proposal to increase the speed of light has been considered but was turned down because of its inter-galactic consequences.

Other maximum cable lengths are (these values are approximate) -

- 100 metres (330 ft) at 500 kbp/s
- 200 metres (650 ft) at 250 kpb/s
- 500 metres (1600 ft) at 125 kbp/s
- 6 kilometres (20000 ft) at 10 kbit/s

Some CAN controllers will handle higher speeds than 1Mbit/s and may be considered for special applications.

CAN connectors

There is no standard at all. Usually, each Higher Layer Protocol (!) defines one or a few preferred connector types. Common types include

- 9-pole DSUB, proposed by CiA.
- 5-pole Mini-C and/or Micro-C, used by DeviceNet and SDS.
- 6-pole Deutch connector, proposed by CANHUG for mobile hydraulics.

Standard vs. Extended CAN

Originally, the CAN standard defined the length of the Arbitration Field to eleven (11) bits. Customer demand forced an extension of the standard. The new format is often called Extended CAN and allows no less than twenty-nine (29) bits in the Arbitration Field. The standards are formally called -

- 2.0A, with 11-bit Arbitration Field only,
- 2.0B, extended version with the full 29-bit (or the old 11-bit, you can mix them) Arbitration Field. A 2.0B node can be active, i.e. it can transmit and receive extended frames, or passive, i.e. it will silently discard received extended frames.

- 1.x refers to the original specification and its revisions. 2.0A and 1.x are compatible.

New CAN controllers today are usually of the 2.0B type. A 1.x or 2.0A type controller will get very upset if it receives a message with 29 arbitration bits. A 2.0B passive type controller will tolerate them; a 2.0B active type controller can both transmit and receive them.

Controllers implementing 2.0B and 2.0A are compatible as long as the controllers implementing 2.0B refrain from sending extended frames!

Sometimes people advocate that standard CAN is "better" than Extended CAN because there are more overhead in the Extended CAN messages. This is not necessarily true. If you use the Arbitration Field for transmitting data, then Extended CAN may actually have a lower overhead than has Standard CAN.

Basic CAN vs. Full CAN

The terms "Basic CAN" and "Full CAN" originate from the childhood of CAN. Once upon a time there was the Intel 82526 CAN controller which provided a DPRAM-style interface to the programmer. Then come along Philips with the 82C200 which used a queue-oriented programming model. To distinguish between the two programming models, people for some reason termed the Intel way as "Full CAN" and the Philips way as "Basic CAN". Today, most CAN controllers allow for both programming models, so there is no reason to use the terms "Full CAN" and "Basic CAN" - in fact, these terms can cause confusion and should be avoided.

Of course, a "Full CAN" controller can communicate with a "Basic CAN" controller and vice versa. There are no compatibility problems.

Message Arbitration

The message arbitration (the process in which two or more CAN controllers agree on who is to use the bus) is of great importance for the really available bandwidth for data transmission. The message arbitration of the CAN protocol consists mainly of comparing bit for bit of the Arbitration Field.

Any CAN controller may start a transmission when it has detected an idle bus. This may result in two or more controllers starting a message (almost) at the same time. The conflict is resolved in the following way. The transmitting nodes monitor the bus while they are sending. If they detect a dominant level when they are sending a recessive level themselves, they will immediately quit the arbitration process and become receivers instead. The arbitration is performed over the whole Arbitration Field and when that field has been sent, exactly one transmitter is left on the bus. This node continues the transmission as if nothing had happened. No time is lost in the arbitration process.

An important condition for this bit-wise arbitration to succeed is that no two nodes may transmit the same Arbitration Field. There is one exception to this rule: if the message contains no data, then any node may transmit that message.

SInce the bus is wired-and and a Dominant bit is logically 0, it follows that the message with the numerically lowest Arbitration Field will win the arbitration.

Error Handling

Error handling is built into in the CAN protocol and is of great importance for the performance of a CAN system. The error handling mainly aims at detecting errors in messages appearing on the CAN bus, so that the Transmitter can retransmit an erroneous message. Every CAN controller along a bus will try to detect errors within a message. If an error is found, the discovering node will transmit an Error Frame, thus destroying the bus traffic. The other nodes will detect the error caused by the Error Frame (if they haven't already detected the original error) and take appropriate action, i.e. discard the current message.

Each node maintains two error counters: the Transmit Error Counter and the Receive Error Counter. There are several rules governing how these counters are incremented and/or decremented. In essence, the a transmitter detecting a fault increments its Transmit Error Counter faster than the listening nodes will increment their Receive Error Counter. This is because there is a good chance that it is the transmitter who is at fault! When any Error Counter raises over a certain value, the node will first become "error passive", that is, it will not actively destroy the bus traffic when it detects an error, and then "bus off", which means that the node doesn't participate in the bus traffic at all.

Using the error counters, a CAN node can not only detect faults but also perform error confinement.

The CAN controller's habit of automatically retransmitting messages when errors have occurred can be annoying at times. There is at least one controller on the market (the SJA1000 from Philips) that allows for full manual control of the error handling.

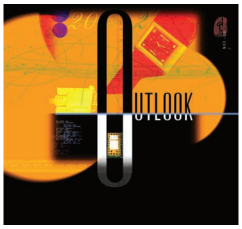
The Layout of a Bit

From a programmer's point of view, each bit on the CAN bus is divided into at least 4 quanta. The quanta are logically divided into four groups or segments -

- the Start Segment
- the Propagation Segment
- the Phase Segment 1
- the Phase Segment 2

The bus levels are sampled at the border between Phase Segment 1 and Phase Segment 2. Most CAN controllers also provide an option to sample three times during a bit. In order to adjust the on-chip bus clock, the CAN controller may shorten or prolong the length of a bit by an integral number of quanta. The maximum value of these bit time adjustments are termed the Synchronization Jump Width, SJW.

Expanding Automotive Electronic Systems



A vast increase in automotive electronic systems, coupled with related demands on power and design, has created an array of new engineering opportunities and challenges.

Gabriel Leen
PEI Technologies

Donal Heffernan
University of Limerick

The past four decades have witnessed an exponential increase in the number and sophistication of electronic systems in vehicles. Today, the cost of electronics in luxury vehicles can amount to more than 23 percent of the total manufacturing cost. Analysts estimate that more than 80 percent of all automotive innovation now stems from electronics. To gain an appreciation of the sea change in the average dollar amount of electronic systems and silicon components—such as transistors, microprocessors, and diodes—in motor vehicles, we need only note that in 1977 the average amount was $110, while in 2001 it had increased to $1,800.[1]

The growth of electronic systems has had implications for vehicle engineering. For example, today's high-end vehicles may have more than 4 kilometers of wiring—compared to 45 meters in vehicles manufactured in 1955. In July 1969, Apollo 11 employed a little more than 150 Kbytes of onboard memory to go to the moon and back. Just 30 years later, a family car might use 500 Kbytes to keep the CD player from skipping tracks.[2]

The resulting demands on power and design have led to innovations in electronic networks for automobiles. Researchers have focused on developing electronic systems that safely and efficiently replace entire mechanical and hydraulic applications, and increasing power demands have prompted the development of 42-V automotive systems.

IN-VEHICLE NETWORKS

Just as LANs connect computers, control networks connect a vehicle's electronic equipment. These networks facilitate the sharing of information and resources among the distributed applications. In the past, wiring was the standard means of connecting one element to another. As electronic content increased, however, the use of more and more discrete wiring hit a technological wall.

Added wiring increased vehicle weight, weakened performance, and made adherence to reliability standards difficult. For an average well-tuned vehicle, every extra 50 kilograms of wiring—or extra 100 watts of power—increases fuel consumption by 0.2 liters for each 100 kilometers traveled. Also, complex wiring harnesses took up large amounts of vehicle volume, limiting expanded functionality. Eventually, the wiring harness became the single most expensive and complicated component in vehicle electrical systems.

Fortunately, today's control and communications networks, based on serial protocols, counter the problems of large amounts of discrete wiring. For example, in a 1998 press release, Motorola reported that replacing wiring harnesses with LANs in the four doors of a BMW reduced the weight by 15 kilograms while enhancing functionality. Beginning in the early 1980s, centralized and then distributed networks have replaced point-to-point wiring.[3]

Figure 1 shows the sheer number of systems and applications contained in a modern automobile's network architecture.

Controller area network

In the mid-1980s, Bosch developed the controller area network, one of the first and most enduring automotive control networks. CAN is currently the most widely used vehicular network, with more than 100 million CAN nodes sold in 2000.

0018-9162/02/$17.00 © 2002 IEEE

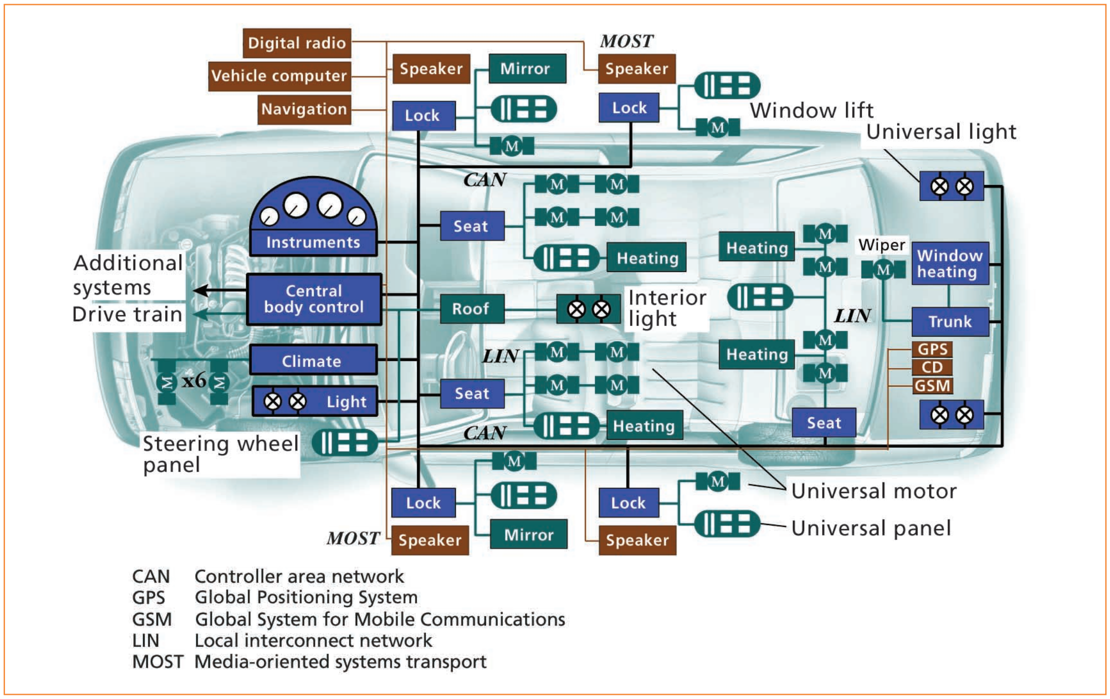


Figure 1. One subset of a modern vehicle's network architecture, showing the trend toward incorporating ever more extensive electronics.

A typical vehicle can contain two or three separate CANs operating at different transmission rates. A low-speed CAN running at less than 125 Kbps usually manages a car's "comfort electronics," like seat and window movement controls and other user interfaces. Generally, control applications that are not real-time critical use this low-speed network segment. Low-speed CANs have an energy-saving sleep mode in which nodes stop their oscillators until a CAN message awakens them. Sleep mode prevents the battery from running down when the ignition is turned off.

A higher-speed CAN runs more real-time-critical functions such as engine management, antilock brakes, and cruise control. Although capable of a maximum baud rate of 1 Mbps, the electromagnetic radiation on twisted-pair cables that results from a CAN's high-speed operation makes providing electromagnetic shielding in excess of 500 Kbps too expensive.

CAN is a robust, cost-effective general control network, but certain niche applications demand more specialized control networks. For example, X-by-wire systems use electronics, rather than mechanical or hydraulic means, to control a system. These systems require highly reliable networks.

Emerging automotive networks

X-by-wire solutions form part of a much bigger trend—an ongoing revolution in vehicle electronics architecture. Multimedia devices in automobiles, such as DVD players, CD players, and digital TV sets, demand networks with extensive synchronous bandwidth. Other applications require wireless networks or other configurations. To accommodate the broad and growing spectrum of vehicle network applications, research engineers are developing many specialized network protocols, including the following.

Domestic Data Bus. Matsushita and Philips jointly developed the Domestic Data Bus (D2B) standard more than 10 years ago, which the Optical Chip Consortium—consisting of C&C Electronics, Becker, and others—has promoted since 1992. D2B was designed for audio-video communications, computer peripherals, and automotive media applications. The Mercedes-Benz S-class vehicle uses the D2B optical bus to network the car radio, autopilot and CD systems, the Tele-Aid connection, cellular phone, and Linguatronic voice-recognition application.

Bluetooth. Bluetooth is an open specification for an inexpensive, short-range (10–100 meters), low-power, miniature radio network. The protocol provides easy and instantaneous connections between Bluetooth-enabled devices without the need for cables. Potential vehicular uses for Bluetooth include hands-free phone sets; portable DVD, CD, and MP3 drives; diagnostic equipment; and handheld computers.

Mobile media link. Designed to support automotive multimedia applications, the mobile media link network protocol facilitates the exchange of data and control information between audio-video equipment, amplifiers, and display devices for such things as game consoles and driver navigation maps. Delphi Packard Electric Systems developed the MML protocol based on a plastic fiber-optic physical layer. Delphi has installed the system in the Network Vehicle, an advanced concept vehicle developed in conjunction with IBM, Sun Microsystems, and Netscape.

Media-oriented systems transport. The applications of MOST, a fiber-optic network protocol with capacity for high-volume streaming, include automotive multimedia and personal computer networking. More than 50 firms—including Audi, BMW, DaimlerChrysler, Becker Automotive, and Oasis SiliconSystems—developed the protocol under the MOST Cooperative (http://www.mostnet.de/main/index.html).

Time-triggered protocol. Designed for real-time distributed systems that are hard and fault tolerant, the time-triggered protocol ensures that there is no single point of failure. The protocol has been proposed for systems that replace mechanical and hydraulic braking and steering subsystems. TTP is an offshoot of the European Union's Brite-Euram X-by-wire project.

Today's vehicle networks are transforming automotive components into truly distributed electronic systems.

Local interconnect network. A master-slave, time-triggered protocol, the local interconnect network is used in on-off devices such as car seats, door locks, sunroofs, rain sensors, and door mirrors. As a low-speed, single-wire, enhanced ISO-9141-standard network, LIN is meant to link to relatively higher-speed networks like CAN. LIN calms fears about security of serial networks in cars. Because LIN provides a master-slave protocol, a would-be thief cannot tap into the network's vulnerable points, such as the door mirrors, to deactivate a car alarm system. Audi, BMW, DaimlerChrysler, Motorola, Volcano, Volvo, and Volkswagen created this inexpensive open standard.

Byteflight. A flexible time-division multiple-access (TDMA) protocol for safety-related applications, Byteflight can be used with devices such as air bags and seat-belt tensioners. Because of its flexibility, Byteflight can also be used for body and convenience functions, such as central locking, seat motion control, and power windows. BMW, ELMOS, Infineon, Motorola, and Tyco EC collaborated in its development. Although not specifically designed for X-by-wire applications, Byteflight is a very high performance network with many of the features necessary for X-by-wire.

FlexRay. FlexRay is a fault-tolerant protocol designed for high-data-rate, advanced-control applications, such as X-by-wire systems. The protocol specification, now nearing completion, promises time-triggered communications, a synchronized global time base, and real-time data transmission with bounded message latency. Proposed applications include chassis control, X-by-wire implementations, and body and powertrain systems. BMW, DaimlerChrysler, Philips, and Motorola are collaborating on FlexRay and its supporting infrastructure. FlexRay will be compatible with Byteflight.

Time-triggered CAN. As an extension of the CAN protocol, time-triggered CAN has a session layer on top of the existing data link and physical layers. The protocol implements a hybrid, time-triggered, TDMA schedule, which also accommodates event-triggered communications. The ISO task force responsible for the development of TTCAN, which includes many of the major automotive and semiconductor manufacturers, developed the protocol. TTCAN's intended uses include engine management systems and transmission and chassis controls with scope for X-by-wire applications.

Intelligent transportation systems data bus. Enabling plug-and-play in off-the-shelf automotive electronics, the intelligent transportation systems data bus eliminates the need to redesign products for different makes. The Automotive Multimedia Interface Collaboration, a worldwide organization of motor vehicle makers, created the specification, which supports high-bandwidth devices such as digital radios, digital videos, car phones, car PCs, and navigation systems. The specification's first release endorses IDB-C (CAN) as a low-speed network and optional audio bus, and two high-speed networks, MOST and IDB-1394b. IDB-1394b is based on the IEEE 1394 FireWire standard.

X-BY-WIRE SOLUTIONS

Today's vehicle networks are not just collections of discrete, point-to-point signal cables. They are transforming automotive components, once the domain of mechanical or hydraulic systems, into truly distributed electronic systems. Automotive engineers set up the older, mechanical systems at a single, fixed operating point for the vehicle's lifetime. X-by-wire systems, in contrast, feature dynamic interaction among system elements.

Replacing rigid mechanical components with dynamically configurable electronic elements triggers an almost organic, systemwide level of integration. As a result, the cost of advanced systems should plummet. Sophisticated features such as chassis control and smart sensors, now confined to luxury vehicles, will likely become mainstream. Figure 2 shows how dynamic driving-control systems have been steadily adopted since the 1920s, with more on the way.[4,5]

Highly reliable and fault-tolerant electronic control systems, X-by-wire systems do not depend on conventional mechanical or hydraulic mechanisms. They make vehicles lighter, cheaper, safer, and more

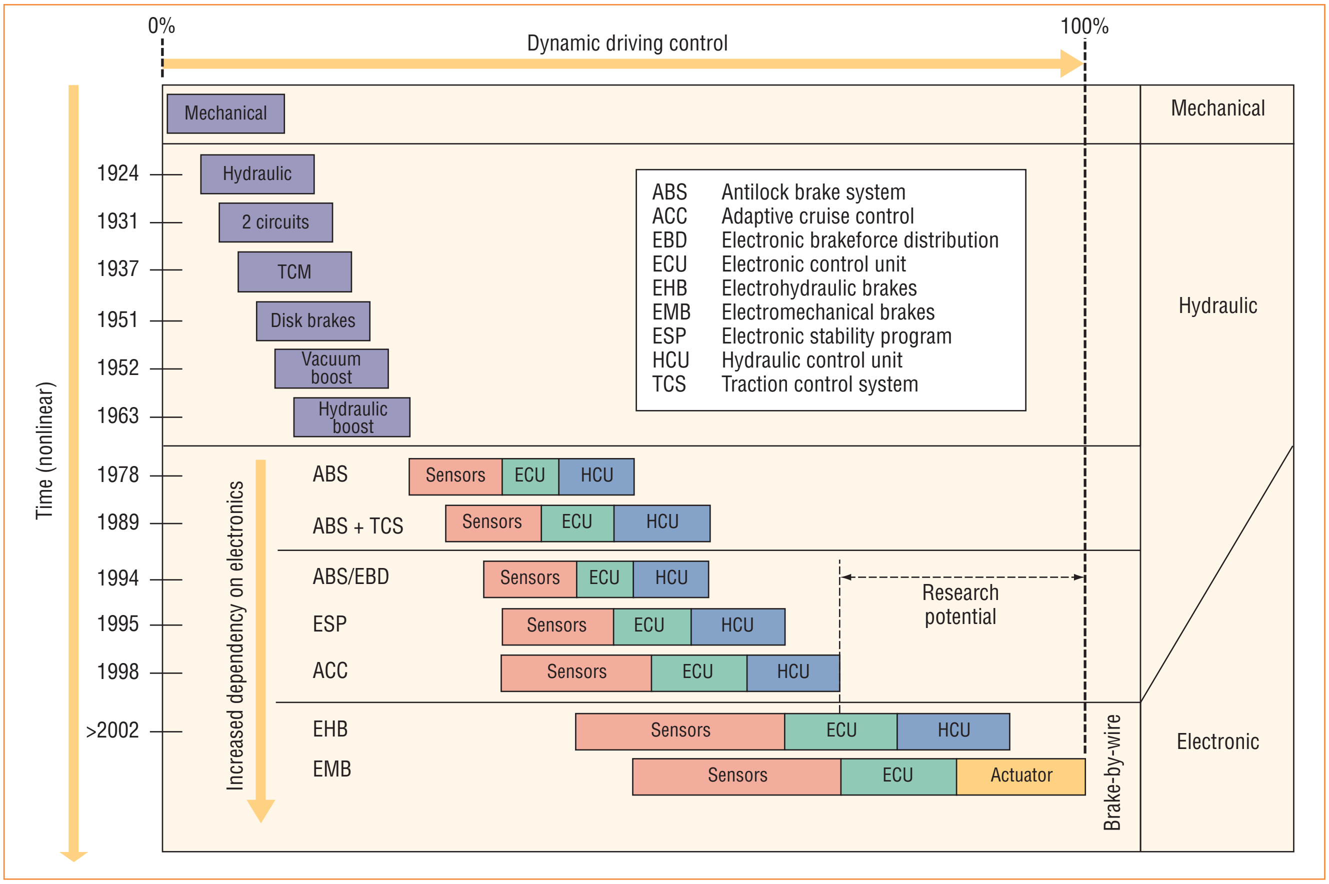


Figure 2. Past and projected progress in dynamic driving control systems. As the cost of advanced systems plummets, sophisticated features are likely to become mainstream components.

fuel-efficient. These self-diagnosing and configurable systems adapt easily to different vehicle platforms and produce no environmentally harmful fluids. Such systems can eliminate belt drives, hydraulic brakes, pumps, and even steering columns.

Indeed, by 2010 one in three new cars will feature electronic steering. X-by-wire steering systems under development will replace the steering column shaft with angle sensors and feedback motors. A wire network will supply the control link to the wheel-mounted steering actuator motors. Removal of the steering column will improve driver safety in collisions and allow new styling freedom. It will also simplify production of left- and right-hand models.

It is natural to add advanced functions to such electronic systems. For example, consider systems that reduce steering-wheel feedback to the driver. In mechanical steering systems, the driver actually feels the vehicle losing control in unstable conditions and can react appropriately. Today, such electronic features as antilock braking may let the vehicle approach or surpass this control-loss edge without providing warning. To accommodate this, X-by-wire systems can include motors on the steering wheel that provide artificial feedback to the driver.

All major automakers are developing prototype or production X-by-wire systems. TRW's electronic power-assisted steering system improves fuel economy by up to 5 percent. Delphi Automotive Systems claims similar improvements from its E-Steer system. Companies such as Bosch, Continental AG, Visteon, Valeo, and most other original equipment manufacturers have either developed or plan to develop X-by-wire technologies and components.

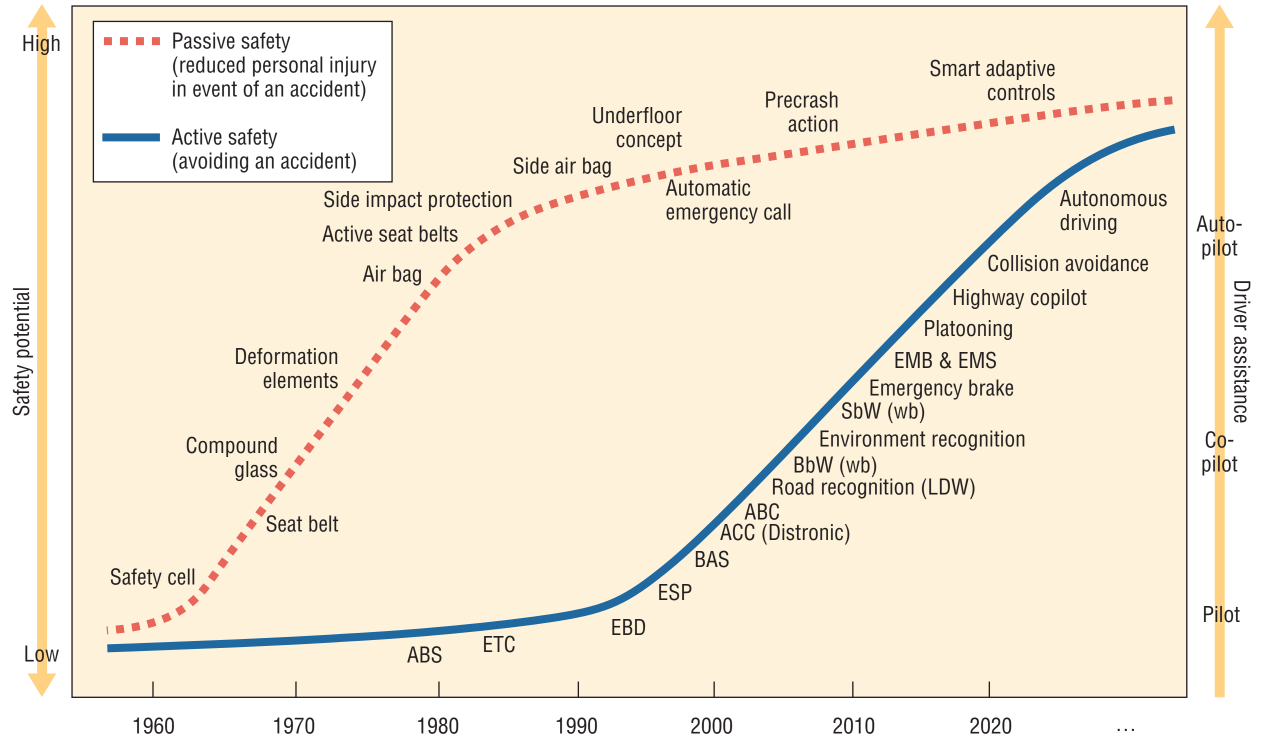
Several protocols are suitable for X-by-wire applications. TTP, for example, is a promising and available protocol geared toward improving driving safety. However, the FlexRay and TTCAN protocols will start to compete with TTP when manufacturers look for more flexibility and lower cost.

Figure 3 shows the past and potential future improvements from active and passive safety systems such as air bags and road-recognition sensors.[6] Advanced electronic systems and the X-by-wire infrastructure will enable most potential active safety improvements.

ELECTRICAL POWER DEMAND

Vehicular battery management systems continuously check the condition of the car's battery, monitoring the charge to ensure the auto will start and have enough power to maintain critical systems. Even with the engine switched off, some systems—real-time clocks, keyless entry and security devices, and vehicle control interfaces such as window switches and light switches—still consume power.

In addition to these conventional electrical systems, emerging applications as diverse as in-car computers and GPS navigation systems consume enough power to raise the total energy load to more than



ABC	Active body control	EBD	Electronic brakeforce distribution
ABS	Antilock brake system	EMB	Electromechanical brakes
ACC	Adaptive cruise control	EMS	Electromechanical steering
BAS	Brake assist system	ESP	Electronic stability program
BbW	Brake by wire	ETC	Electronic traction control
CA	Collision avoidance	SbW	Steer by wire
DbW	Drive by wire	(wb)	with mechanical backup

Figure 3. Past and future active and passive safety systems. Advanced electronic systems and the X-by-wire infrastructure will enable active safety improvements.

2 kW. If historical trends continue, internal power demand will grow at a rate of 4 percent a year. Conservative estimates put the average electrical power requirements for high-end vehicles at 2.5 kW by 2005.[7] These increases place strains on conventional power equipment. For example, at a 3-kW load, bracket-mounted, belt-driven alternators generate unpleasant noises and require liquid cooling.

Table 1 shows some anticipated electrical loads for key emerging systems.[8] Analysts expect the loads to reach the listed levels by 2005. Electromechanical valves that will replace the camshaft and inlet and exhaust valves offer one exception—they probably won't be produced until 2010.

Given the benefits they offer, such systems and their greater power loads are necessary. Electromechanical valves, for example, should provide a 15 percent improvement in fuel consumption. Preheated catalytic converters will decrease exhaust emissions by 60 to 80 percent.

THE 42-V SOLUTION

To meet the increasing demand for power, a beltless engine with an integrated alternator-starter on the flywheel operating at a 42-V potential offers the most promising proposed solution. The motive for the new 42-V system is clear: 79 percent of the energy entering a conventional engine does not make it to the driveline.[2] The standard Lundell claw-and-rotor alternator is itself only 30 percent efficient at high speeds and 70 percent efficient at low speeds. Thus, generating a watt of electrical power requires about 2 watts of mechanical power, with the lost watt turned into heat.

The integrated system is expected to be 20 percent more efficient, providing a benefit of roughly 0.2 km/liter, or 0.4 mpg. Its "lite hybrid" alternator-starter will operate the vehicle in start-and-stop mode, in which the engine can be restarted in 200 ms for even more fuel savings. In addition, removal of the front-end accessory drive—running the alternator and power-steering pump—will mean enhanced car styling. The new 42-V systems are expected in new autos by 2003.

Within the electrical system, boosting the voltage proportionally reduces the required current for a given delivered power. Smaller currents will use smaller and lighter-gauge cables, allowing an expected 20 percent reduction in cable bundle size. Further, the carrying capacity of semiconductor switches for electrical currents relates directly to silicon area size, while operational voltage levels are a function of device thickness and doping profile. With less silicon area required, these systems

Table 1. Predicted electrical loads of advanced electronic systems.

System	Peak load	Average load
Electromechanical valves	2,400	800
Water pump	300	300
Engine cooling fan	800	300
Power steering (all electric)	1,000	100
Heated windshield	2,500	200
Preheated catalytic converter	3,000	60
Active suspension	12,000	360
Onboard computing, navigation		100
Total average		**2,220**

will achieve a significant cost reduction in solid-state load-switching devices.[1]

The 42-V systems will require a 36-V battery and produce a maximum operating level of 50 V, with a maximum dynamic overvoltage of 58 V. Engineers regard a 60-V limit as the safe maximum for cars; greater voltages can generate shocks.[9]

Despite the obvious advantages of 42-V systems, challenges loom. Transition costs—reengineering of products and production processes—will be extremely high due to the legacy of a half century of 12-V systems. The upgrading of service and maintenance equipment will provide other obstacles. Still, annual power consumption increases of 4 percent will simply overload present-day 14-V systems, making 42-V alternatives inevitable.

Reducing wiring mass through in-vehicle networks will bring an explosion of new functionality and innovation. Our vehicles will become more like PCs, creating the potential for a host of plug-and-play devices. With over 50 million new vehicles a year, this offers the potential for vast growth in automotive application software—much like that of the PC industry over the past decade.

On average, US commuters spend 9 percent of their day in an automobile. Introducing multimedia and telematics to vehicles will increase productivity and provide entertainment for millions. Further, X-by-wire solutions will make computer diagnostics a standard part of mechanics' work. The future could even bring the introduction of an electronic chauffeur. ■

Acknowledgments

We thank the Byteflight Group, DaimlerChrysler, Delphi Automotive, the FlexRay Group, Siemens, Motorola, PEI Technologies, TTTech, and the University of Limerick for their assistance.

References

1. J.M. Miller et al., "Making the Case for a Next-Generation Automotive Electrical System," MIT/Industry Consortium on Advanced Automotive Electrical/Electronic Components and Systems, http://auto.mit.edu/ consortium (current Dec. 2001).
2. W. Powers, "Environmental Challenges, Consumer Opportunities," Auto.com, http://www.auto.com/travcity99/wpowers_aug5.htm (current Dec. 2001).
3. G. Leen, D. Heffernan, and A. Dunne, "Digital Networks in the Automotive Vehicle," *IEE Computer and Control Eng. J.*, Dec. 1999, pp. 257-266.
4. "Electronic Brake Management," *ALex Current Factbook*, BMW Research and Development, http://www.bmwgroup.com/e/index2.shtml?s50&0_0_www_bmwgroup_com/4_news/4_4_aktuelles_lexikon/4_4_aktuelles_lexikon.shtml (current Dec. 2001).
5. A. van Zanten et al., *ESP Electronic Stability Program*, Robert Bosch GmbH, Stuttgart, Germany, 1999.
6. T. Thurner et al., *X-By-Wire: Safety Related Fault-Tolerant Systems in Vehicles*, Document No. XBy-Wire-DB-6/6-25, X-by-Wire Consortium, Stuttgart, Germany, 1998.
7. J.M. Miller, "Multiple Voltage Electrical Power Distribution Systems for Automotive Applications," *Proc. 31st Intersociety Energy Conversion Conf.*, IEEE Press, Piscataway, N.J., 1996, pp. 1930-1937.
8. J.G. Kassakian, "Automotive Electrical Systems: The Power Electronics Market of the Future," *Proc. Applied Power Electronics Conf. and Exposition* (APEC 2000), IEEE Press, Piscataway, N.J., 2000, pp. 3-9.
9. Institute of the Motor Industry, "Volting Ahead on Power Systems," http://www.just-auto.com/features_detail.asp?art=307 (current Dec. 2001).

***Gabriel Leen** is a technical researcher at PEI Technologies, University of Limerick, Ireland. His research interests include in-vehicle networks, formal verification of vehicle network protocols, and automotive computing. Leen has several years' experience in automotive electronic system design. He received a research MEng from the University of Limerick and is currently completing a PhD in automotive networking design. Leen is a member of the Institution of Engineers of Ireland. Contact him at gabriel.leen@ul.ie.*

***Donal Heffernan** is a lecturer in computer engineering at the University of Limerick, Ireland. His research interests are real-time embedded system design and reliable protocols for distributed control networks. He received an MS in electrical engineering from the University of Salford, UK. Heffernan is a member of the Institution of Engineers of Ireland. Contact him at donal.heffernan@ ul.ie.*